

C Programs and Documentation

by Nicholas Regan

FILE	12125_NICHOLAS_REGAN_C_PROGRAMS_AND_DOCUMENTATION_157877_2045900294.DOCX (286.22K)		
TIME SUBMITTED	25-FEB-2019 02:18PM (UTC+0000)	WORD COUNT	2628
SUBMISSION ID	101180877	CHARACTER COUNT	12496

Title: Programmes in C and Documentation

Module: Software Development

ACCA4017

Course: HND Electronics and Electrical Engineering

University of Wales Trinity St David

By: Nick Regan

Student No: 1803871

Date: 25.2.19

Declaration

The work of this assignment which has been completed under the supervision of
Gaynor Thomas:

I declare that this assignment is the product of my own work and has not been
submitted for any other degree etc.

Signed: N.S.B.

Date: 25/2/2019

Contents

Description	Page Number
Introduction	1
1. C Programming	2
1.1 Variables	2
1.2 For-Loops	6
1.3 Arrays	7
2. Exercise 1	10
3. Exercise 2	11
4. Exercise 3	14
Reference List	16
Table List	16
Figure List	16

Introduction

This assignment is based on C Programming and discusses variables, explaining what they are and why they are necessary in writing programmes. It will then discuss for-loops explaining what they do and why they could also be called a counting-loop. The first section will finish by discussing 'arrays' their features and what they can be used for.

This is followed by examples of exercises in programme writing showing:

- Problems in completion
- Clear analysis of the solution.
- C code listing with annotations
- Screenshots showing that the solutions are workable.

1. C Programming

C programming was created in 1972 by Dennis M Richie in the U.S.A. Previous to this in the 1970's there was a programming method known as 'B' so Richie's updated method was called 'C', [1 p8]. McGrath points out that "C remains popular because it is versatile and efficient." [1 p8]. Also, in Perry and Miller's opinion, it is the best programme to start learning on as although it is basic, they state that "C is a programming language rich in its capabilities." [2 p5].

Basic computer programming using switches and rewiring begun in the 1940's when electric computers first appeared. In the 1950's Professor John von Nuemann pioneered a computer programming language and introduced the concept of computers making 'if-then' decisions and repeating loops. [3 p21].

C Programming was improved and updated over the years and became C++ and C#. These programming languages are based on C but obviously more complicated and can carry out advanced tasks.

When writing programmes in C, a programme writer needs a C compiler, to build or compile the programme. It is a software package that writes, edits and debugs the programme without switching software. This is known as Integrated Development Environment.

1.1 Variables

In C and most programme languages, data that is created has to be **stored** in the computer. Gookin explains, "The computer storage keeps the items, the containers themselves are called 'variables'. [3 p71]. All computer programming includes variables. Variables are the "basic component of all computer programming." [3 p71].

Gookin continues by explaining that variables as places to store information, [3 p71]. Information that has been created, gathered needs to be kept for reuse in the future. So variables are these storage places. Perry and Miller explain in simple terms, it's like "a box in the computer's memory that holds a number or a character." [2 p41].

C programmes use two types of values, immediate and variable. Immediate is the one specified in the source code, and stays the same, (constant). The value in variable can change. Hence called 'variable'.

There are several types of variables which hold different types of data and values. Gookin gives an example in simple terms, saying that an animal variable type would contain horses and cows and a flower variable type would contain roses and lilies. He explains that "C language variables work along these lines, with specific values assigned to **matching** types of variables." [3 p73]. So a 'flower' value cannot be assigned to an 'animal' variable.

In computer programming there are several types of data:

Characters... there are 256 'characters' a computer can use. For example letters, numbers, punctuation marks = + space bar. If more than a single character is used it is then called a string. Values put into character variable types are put into apostrophes if they are single and quotation marks if they are a string.

Integers... are numbers, whole numbers with no decimal points.

Floating point numbers...numbers with decimal points

There are 4 basic variable types:

Data type	Description	Example
char	A single byte, capable of storing just one character	'A' or '^'
int	An integer whole number	100
float	A floating point number correct to six decimal places	0.123456
double	A floating-point number correct to ten decimal places	0.123456789

Table 1: C Language Basic Variable Types used in C Language with Examples

Variables must be given a name. It is the programmer's responsibility to name each one. Each must have a different name and it can have 1 to 31 characters in the name. They must always start with a letter and then be followed with other letters, underscores or numbers. There cannot be any spaces or punctuation marks or arithmetical operators (+). They can have mixed case, but usually only lower case letters are used. Perry and Miller point out that the name cannot contain C keywords, functions or commands. [2 p45]. It is useful to give the variable a name that is linked to the topic so that it can be remembered, recognised and identified in the future.

There are three stages to creating a variable;

- Defining
- Assigning
- Using

Defining (sometimes known as declaring) a variable means stating its type followed by a space then the name [3 p21.] This is always done at the beginning of a function after the initial curly bracket.

For example:

```
{  
  
char x;
```

The data type is char and the name is 'x'. The semi colon indicates the end of the name.

Assign a value to the variable by using an = sign after the name. (An assignment operator). This value can be changed in the future, (variable).

For example:

```
x = 3.
```

Below is an example from Gookin, [3 p74], of the defining of four variables and their values:

Working with Variables

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

```
Int main ( )
```

```
{
```

```
Char c;
```

```
Int i;
```

```
Float f;
```

```
Double d;
```

```
C = 'a'
```

```
I = 1
```

```
F = 19.0;
```

```
d = 2000.009
```

```
}
```

Figure 1: Illustration of Defining names and Values to Variables

Variables are very important in programming as using them allows changes and flexibility. "The variable is truly the heart of any programming language by allowing you to code flexibility into your programs." [3 p76]. The flexibility is possible as the values can be changed each time a programme is executed and also variables save something that can be changed due to a calculation.

1.2 For-loops

Sometimes it is necessary and useful to use 'loops' in C programmes. A loop means the same section of code repeated a few times and then to finish with a conclusion or result. Loops are for repetitive tasks like adding up the number of applicants, or adding up how many items are sold, or adding up the total price of a number of items. There is a 'while loop' and a 'do-while loop' both which continue looping while the condition is true, (stays the same). Perry and Miller explain how a while loop can be useful by giving an example of recording customer's purchases as they come into the shop to buy. [2 p131]. It will repeat the process every time a customer purchases an item. (the true condition).

However another type of C loop is called a 'for-loop'. "With for-loops you can **specify** exactly how many times you want to loop." [2 p 131]. So using Perry and Miller's same example scenario as customers purchasing items, if it was required to total the purchases for **one** particular day, when the while loop recorded 100 customers, a for-loop could then be used as it would be known exactly how many times to loop, (100 times) to total the purchases, before stopping. So, a for-loop allows control as it can count. It can count up or down, so can increment or decrement. For this reason it is sometimes known as a 'counting loop'.

There are three stages of a for-loop:

initialisation, which is the start of the loop, (when to begin the loop). Th

exit_condition, which is the test upon which the loop stops as long as the exit condition stays true. (This is the most important part as otherwise it would loop endlessly.)

repeat_each, execute once every repeat.

Below is an illustration of this process:

for (initialisation; exit_condition; repeat_each)

for (x=0; x<10; x=x+1)

Figure 2: Illustration of the Three Stages of a For-loop

Explanation of Figure 2:

$x=0$ (start counting at 0 (not 1) C always starts counting at 0)

$x \leq 10$ (repeat the loop as long as the value remains less than 10, so that at 9 it will stop... therefore a count of 10)

$x=x+1$ (Value increases by 1 each loop repeat).

Another example of the use of a for-loop could be to identify days in the year that were hottest, i.e. above a certain temperature. There would be 365 entries (days in the year) so it could be set to count 365 times, and then could be set to identify temperature measurements more than 18 degrees Celsius.

1.3 Arrays

An array is a special variable. It is actually a list of variables of the same data type, so it can store lots of items of data instead of a single item stored by a normal variable. [1 p34]. Arrays are lists of values of the same data type and each data type has its own array.

Perry and Miller [2 p193], explain simply:

An array of characters is a list of characters that has a name.

An array of integers is a list of integers that has a name.

An array of floating point values is a list of floating pint values that has a name.

Gookin describes an array by picturing it as "a series of cubby holes into which you stick different values." [3 p164].

The list of char variables stored in a character array is known as a 'string'. Character arrays hold strings in memory. A string is always ended with a terminator called a null zero, which C will perform automatically. It is invisible but takes up a space. It is written '\0'. Strings are contained in double quotes, "string", whereas single characters are contained in apostrophes.

Each array has a number of values, known as elements. Instead of giving each element a different name, they are given a subscript (numbers starting from 0)

Each element has an assigned subscript, (see Table 2). Each element in the array has to be the same data type.

To tell C that you need a character array, (rather than a single character variable), it is declared slightly differently. [2 p52]

It is the same as a normal variable but with the data type, the name, followed by square brackets containing a size. The size is calculated by finding the longest word that would occur in the list of the values to be entered. If it was an array named 'days' and the values to be entered were days of the week it would be 'Wednesday' which has 9 letters, plus the null zero which always follows and is put in by C automatically. So the number in the square brackets would therefore be [10].

Each individual piece of the array is called an 'element' [1 p53]. Each array element is also assigned a subscript starting with 0. The subscripts are numbers and are put in square brackets. Each number refers to an element in the array.

Subscripts
↓

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
W	e	d	n	e	s	d	a	y	\0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
M	o	n	d	a	y	\0			

↑
String terminator

Table 2: Illustration of an Array with String Data, showing Subscripts and String Terminator

Values can be assigned to arrays by:

Assignment: which are the initial entries at the start which are known in advance.

User data entry: an unknown value entered by an outside use, such as customers placing orders or recording a result.

Disk files: volumes of data contained on one disk.

An example of the use of an array could be an analysis of the amount of weight lost by a dieter over a ten week period. The first six weeks of the amount of pounds lost would be entered initially, leaving the user to enter the remaining four weeks. A loop can copy each score so the average can be worked out, or the week where the most weight was lost, or a weight loss amount for a particular week.

Another example could be to record a list of customers, by using their ID number and then another array with the amount of purchases they have made in monetary value.

These arrays can be linked, (parallel arrays), i.e. customer ID number, to the same subscript in another array to say how much money they owe. A for-loop can be used to locate customer's ID and link it to the amount owed. A for-loop can go through each one to see who owes more than £100 if that condition is set as true. [2 p205]

Other examples include:

Laboratory test results

Weather data, such as inches of rainfall or temperatures.

2. Exercise 1

1.1 Identify the problem: (break down into smaller problems, write individual stages)

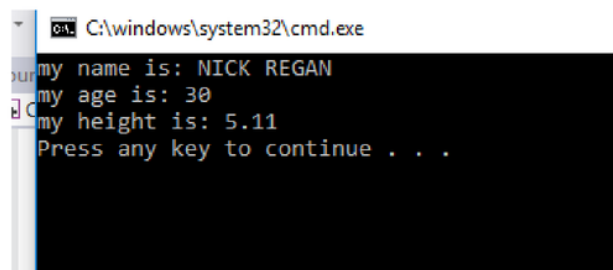
1. Input name (char strings as it is multiple characters)
2. Input age (int as it is a whole number)
3. Input height (float as it is a decimal)
4. Print values on screen

1.2 Annotated code

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

```
2
3 void main()
4 {
5
6     int myage;
7     char myname[5] = { 'N', 'I', 'C', 'K' }; //string = multiple characters
8     char mysurname[6] = { 'R', 'E', 'G', 'A', 'N' };
9     float myheight;
10
11     myage = 30;
12     myheight = 5.11;
13
14
15     printf("my name is: %s %s\n\n", myname, mysurname);
16     printf("my age is: %d\n", myage);
17     printf("my height is: %.2f\n", myheight);
18 }
19
```

1.3 Testing



```
C:\windows\system32\cmd.exe
my name is: NICK REGAN
my age is: 30
my height is: 5.11
Press any key to continue . . .
```

Result: program ran as expected correct values printed on screen

3. Exercise 2

2.1 Identify the problem: (break down into smaller problems, write individual stages)

1. User enters first number (float as maybe decimal)
2. Store in base
3. User enters second number
4. Store in height
5. Multiply base by height and divide by 2
6. Store in Tarea
7. Print Tarea on screen
8. Enter third number
9. Store in radius
10. Multiple PI by radius by radius
11. Store in Carea
12. Print on screen
13. Subtract Tarea from Carea
14. Store in Sarea
15. Print on screen

2.2 Annotated code

```
Project3 (Global Scope) main()
1  #include <stdio.h>
2  #define PI 3.142
3
4  void main()
5  {
6
7      float base, height, radius, Tarea, Carea, Sarea; // variouables to hold user input and calculation
8
9      printf("please enter triangle base ");
10     scanf("%f", &base); // user enters base
11     printf("please enter triangle height ");
12     scanf("%f", &height);
13     Tarea = base * height / 2; // multiply base by height and divide by 2
14     printf("the area of the triangle is %.2f\n", Tarea); //print area of trinagle
15
16     printf("please enter the circle radius ");
17     scanf("%f", &radius);
18     Carea = PI * radius * radius;
19     printf("the area of the circle is %.2f\n", Carea);
20
21     Sarea = Carea - Tarea;
22     printf("the area of the shaded part is %.2f", Sarea);
23
24
25 }
```

2.3 Testing

2.3.1 normal values

```
please enter triangle base 7
please enter triangle height 5
the area of the triangle is 17.50
please enter the circle radius 12
the area of the circle is 452.45
the area of the shaded part is 434.95
C:\Users\nickr\source\repos\Project3\Debug\Project3.exe (process 14140) exited with code 0.
Press any key to close this window . . .
```

Result: program ran as expected and calculated the area of the triangle, circle and shaded part correctly.

2.3.2 normal values (but negative)

C:\Users\mickr\source\repos\Projects\Debug\Projects

```
please enter triangle base -5  
please enter triangle height -7  
the area of the triangle is 17.50  
please enter the circle radius -5
```

Result: program failed after this point

2.3.3 letters

C:\Users\mickr\source\repos\Projects\Debug\Projects

```
please enter triangle base a
```

Result: program failed

4. Exercise 3

3.1 Identify the problem: (break down into smaller problems, write individual stages)

1. User enters 5 prices
2. Store in products
3. Add prices to get total price
4. Store in total1
5. Multiply total price by vat
6. Store in total2
7. Add total1 and total2
8. Store in total3

3.2 Annotated code

```
#include<stdio.h>
#define VAT 0.20
void main()
{
    float product[5];
    int i;
    float total, total2, total3;

    for (i = 0; i < 5; i++)
    {
        printf("\tPlease enter the price: ");
        scanf("\t%f", &product[i]);
    }

    total = product[0] + product[1] + product[2] + product[3] + product[4];
    printf("\tThe total before VAT is: %.2f\n", total);
    total2 = total * VAT; //total x VAT
    total3 = total + total2;
    printf("\tThe total after VAT is: %.2f\n\n", total3);
}
```

3.3 Testing

3.3.1 normal prices

```

Please enter the price: 7
Please enter the price: 5.50
Please enter the price: 2.99
Please enter the price: 3.00
Please enter the price: 0.99
The total before VAT is: 19.48
The total after VAT is: 23.38

\\Users\nickr\source\repos\Project7\Debug\Project7.exe
Press any key to close this window . . .

```

Result: Program ran as expected and calculated the total before vat and after vat.

3.3.2 normal values (but negative)

```

Please enter the price: -7
Please enter the price: -3.5
Please enter the price: -0.99
Please enter the price: -4.3
Please enter the price: -5

```

Result: program failed

3.3.3 letters

```

Please enter the price: a
Please enter the price:
Please enter the price:
Please enter the price:
ice: The total before VAT is: -536870880.00
The total after VAT is: -644245056.00

\\Users\nickr\source\repos\Project7\Debug\Project7.exe (process 9868) exited with code 0.
Press any key to close this window . . .

```

Result: program failed

Reference List

- [1] M. McGrath, C Programming in Easy Steps, 3rd Ed., Southam, Warwickshire, In Easy Steps, 2009.
- [2] G. Perry, D. Miller, C programming Absolute Beginner's Guide, 3rd Ed., Indiana USA, Que Publishing, 2013
- [3] D. Gookin, Beginning Programming with C for Dummies, Hoboken, New Jersey, John Wiley & Sons, Inc. 2014

Table List

Table 1: Basic C language Variable Types, created by the author of this assignment using information from M. McGrath, C Programming in Easy Steps, 3rd Ed., Southam, Warwickshire, In Easy Steps, 2009, p8.

Table 2: Illustration of an Array with String Data, showing Subscripts and String Terminator,, created by the author of this assignment using information from G. Perry, D. Miller, C programming Absolute Beginner's Guide, 3rd Ed., Indiana USA, Que Publishing, 2013, p53.

Figure List

Figure 1: Illustration of Defining Variables and Adding Values: taken form D. Gookin, Beginning Programming with C for Dummies, Hoboken, New Jersey, John Wiley & Sons, Inc. 2014, pp. 74-75.

C Programs and Documentation

GRADEMARK REPORT

FINAL GRADE

/100

GENERAL COMMENTS

Instructor

Part A

Variables - Quite comprehensive. Be careful when copying and pasting - looks like Word has capitalised some parts of your code eg `Int main()` `Char c` etc which would break if run. You could also talk about the different memory allocated to different types - different types take up different memory lengths. 4.5/5

Counting Loops - Be careful with your examples, greater than or equal to is written `<=` (not `=>` which will not run) 4.5/5

Arrays - A bit overexplained if anything! I'd like to see a code example here 4.5/5

Part B

Exercise 1

Code-Annotation - You have not allowed the user to input values but have hard-coded them instead, therefore the program is a bit limited. With the user entering values you could then go on and validate input eg for age 2/5

Documentation - Reasonable breakdown, but testing is limited because the program is limited 2/5

Exercise 2

Code-Annotation - A working solution, but you need to think about validation here - what if negative numbers were entered? What if the triangle was a lot bigger than the circle? 2.5/5

Documentation - It is good that you have entered impossible values such as letters - I don't mind these failing, but the negative numbers problem should have been addressed in the code. This is the whole point of testing - test, adapt the code, and then show that the program can cope with out of range values. 3/5

Exercise 3

Code-Annotation - I'm a bit confused why you've done a loop for the entry, but not for adding the values - this is inefficient. Also, do not include anything on scanf other than the marker eg %f. It doesn't make sense to include a tab on a scanf, because scanf is input, not output. You also need to account for negative values being entered here. 2.5/5

Documentation Breakdown is OK other than not considering the range of possible values. Same comment for testing as exercise 2 - negative numbers should have been addressed 3/5

Exercise 4 - not attempted? You could have included the breakdown even if the code was too difficult?

Code-Annotation 0/10

Documentation 0/10

Total: 28.5/65 (44%)

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19
