Lesson 1

Elementary C Programs

In this lesson, you will learn about:

- Comments
- The `#include`, and header files such as `<stdio.h>`
- The `main{}` function
- The standard functions `printf()`, and `scanf()`
- The elementary numerical data types:
  - `int` (integer)
  - `float` (floating point)
  - `char` (character)
- Declaring variables, with meaningful variable identifiers
- Initializing variables as they are declared
- The use of the conversion specifiers: `%d`, `%f`, `%c`
- How to get data into a program, and how to print data to the screen
- Some C operators:
  - (`+`, `-`, `*`, `/`) arithmetical
  - (`=`, `==`) "takes the value of", and "equal to"
  - (`%`, `&`) modulus, and "address of"

Important Notes:

- **You should type in all the programs in this handout, and run them more than once with different data**
- **You should read, and understand everything in this handout, the material in it forms the basis of the quizzes**
- **If you don't understand something; ask me to explain.**

Some useful definitions

Computer program:

A simple program, is a piece of computer software that takes data in from a user, does something with that data, and returns data to the user.

Source code:

Is the plain text C program that can printout, and save to a floppy disk.

Compiler:

A compiler, is a piece of software that translates the source code, into a program that the computer can run.

Comments:

Comments are sections of the source code bracketed by the `/*` and `*/`, or following the `//`, which the compiler ignores. They are used for putting short explanations of the program into the source code. You can also comment-out parts of the program that don't work properly.

Data types:

In computer programs, each piece of data must have a type. This tells the compiler how much memory each piece of data needs, and what operations can be done with the piece of data. The elementary data-types in C are:

`int` (integer) All the whole numbers, positive and negative
`float` (floating point) All the real numbers -- 3.14, 7.99
`char` (character) All the keyboard characters -- *, \, s, 2, ?, >

Variable identifiers:

In programming, each piece of data is called a variable, (because the value it holds can be changed) and each variable must have a name. These names are called variable identifiers, and they are names made up by the programmer, and can be anything you want, as long as they begin with a letter. It is also good practice to choose names that give a clue as to what they contain. It's clear what the variable `student_name`, and `social_security_number` might contain, but `name`, and `number` thought both legal variable identifiers, don't tell you as much.

Standard functions:

All programming languages have standard functions. These are little programs that do specific jobs, you will soon learn how to use two: `printf()`, and `scanf()`.

Statement terminator:

In C, each statement must end with either the semi-colon (`;`), or curly-bracket (`}`). More on this later.

The `main` function:

In C, the main function is the whole program. Below is a basic (but complete) main function.

```
void main(void)   // The program starts here
{

}   // The program ends here
```

Conversion specifiers:

In C, when we read in data, or print it out, we must tell the standard functions how to we want the data treated. For this we use conversion specifiers, each data type has its own conversion specifer:

`int` uses `%d`
`float` uses `%f`
`char` uses `%c`

Escape sequences:

In C, there are many escape sequences, the ones you will use the most are the: `\n` (backslash - n), and `\t` (backslash - t). The `\n` creates a new line, and `\t` is the tab, when each are used within the standard function `printf()`.

ASCII Table (American Standard Code for Information Interchange)

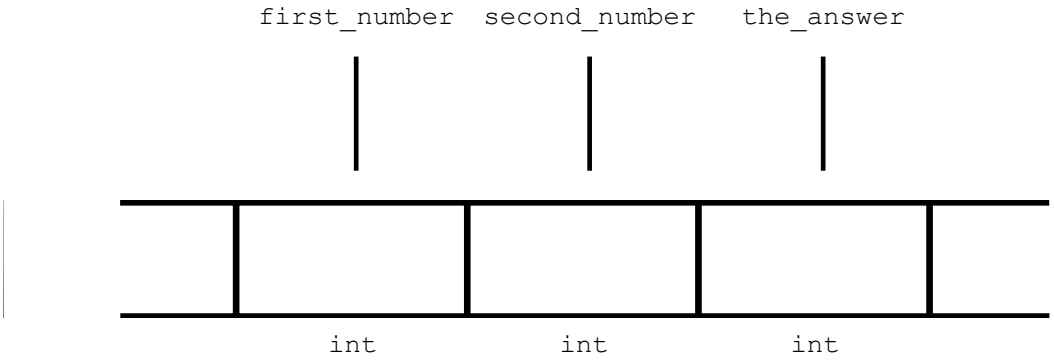
The values below, are how the computer represents characters. It doesn't see any difference between the character `'#'`, and `'a'`.

32 =	33 = !	34 = "	35 = #	36 = $
37 = %	38 = &	39 = '	40 = (	41 =)
42 = *	43 = +	44 = ,	45 = -	46 = .
47 = /	48 = 0	49 = 1	50 = 2	51 = 3
52 = 4	53 = 5	54 = 6	55 = 7	56 = 8
57 = 9	58 = :	59 = ;	60 = <	61 = =
62 = >	63 = ?	64 = @	65 = A	66 = B
67 = C	68 = D	69 = E	70 = F	71 = G
72 = H	73 = I	74 = J	75 = K	76 = L
77 = M	78 = N	79 = O	80 = P	81 = Q
82 = R	83 = S	84 = T	85 = U	86 = V
87 = W	88 = X	89 = Y	90 = Z	91 = [
92 = \	93 =]	94 = ^	95 = _	96 = `
97 = a	98 = b	99 = c	100 = d	101 = e
102 = f	103 = g	104 = h	105 = i	106 = j
107 = k	108 = l	109 = m	110 = n	111 = o
112 = p	113 = q	114 = r	115 = s	116 = t
117 = u	118 = v	119 = w	120 = x	121 = y
122 = z	123 = {	124 = |	125 = }	126 = ~

Program variables as boxes in computer memory

When you declare variables in a computer program, it can be helpful to think of each variable as a box in the computer's memory, that you can put data into. Each box has a name, and a data type, the name is what the programmer uses to refer to that particular box, and the data type restricts the kind of data that can be stored in the box.

Below is a pictorial representation of how the computer's memory is used in the program `add_two.c`



What we have done, is to tell the compiler to:

- Allocate three areas in memory
- That these three areas will store integers
- What the names of these three areas will be

Designing programs for an "eight-year-old"

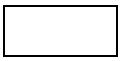
I believe, that simple computer programs can be designed so that an average eight-year-old child could carry out the instructions written down on paper. Below in the "eight-year-old" design for the program `add_two.c`

---

declare integers "first_number", "second_number", "the_answer"

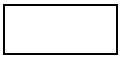
ask for a number

put the number in the box "first_number"



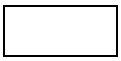
ask for a number

put the number in the box "second_number"



add the number in the box "first_number" and the number in the box "second_number"

and put the result in the box "the_answer"



tell me the number in the box "the_answer"

Program (`add_two.c`)

```
/*
  Program "add_two.c"

  This program asks the user for two integers.
  It adds them together, then prints the result to the screen.
*/

#include <stdio.h>

void main(void)
{
int first_number = 0,
    second_number = 0,
    the_answer = 0;

  printf("Please enter an integer: ");
  scanf("%d",&first_number);

  printf("Please enter an integer: ");
  scanf("%d",&second_number);

  the_answer = first_number + second_number;

  printf("Adding %d and %d ",first_number,second_number);
  printf("results in: %d \n",the_answer);
}
```

An exploded view of the program "add_two.c"

```
/*
  Program "add_two.c"
  Written by: Joe Dorward
  Date 03/09/00

  This program asks the user for two integers.
  It then adds them together, then prints the result to the screen.
*/
```

The text between /*, and */ is a comment.
Comments are ignored by the compiler.
Programmers use comments to explain program code, and to leave messages to other programmers who will read the program in the future.

A compiler directive to "include" a file in the program source code.

The name of the "header file" to be included in the program source code.

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

```
void main(void)
{
```

The start of the program.

The data type of the variable(s) to be declared, int is an abreviation of integer.

The list of variable(s) being declared.
These names can be anything, and are made up by the programmer.
These variables, are "initialized" (given an initial value) to 0.

```
int  first_number = 0,
     second_number = 0,
     the_answer = 0;
```

The standard function for writing to the screen.
This function has nothing to do with storing data in memory.

A prompt, it's how we tell the user what to do. It's just a line of text.
The printf() will print every thing between the quote marks to the screen.
The '\n' is the newline character, this causes the printf() to take a new line.

```
  printf("\n Please enter an integer: ");
```

The semi-colon is the statement terminator.

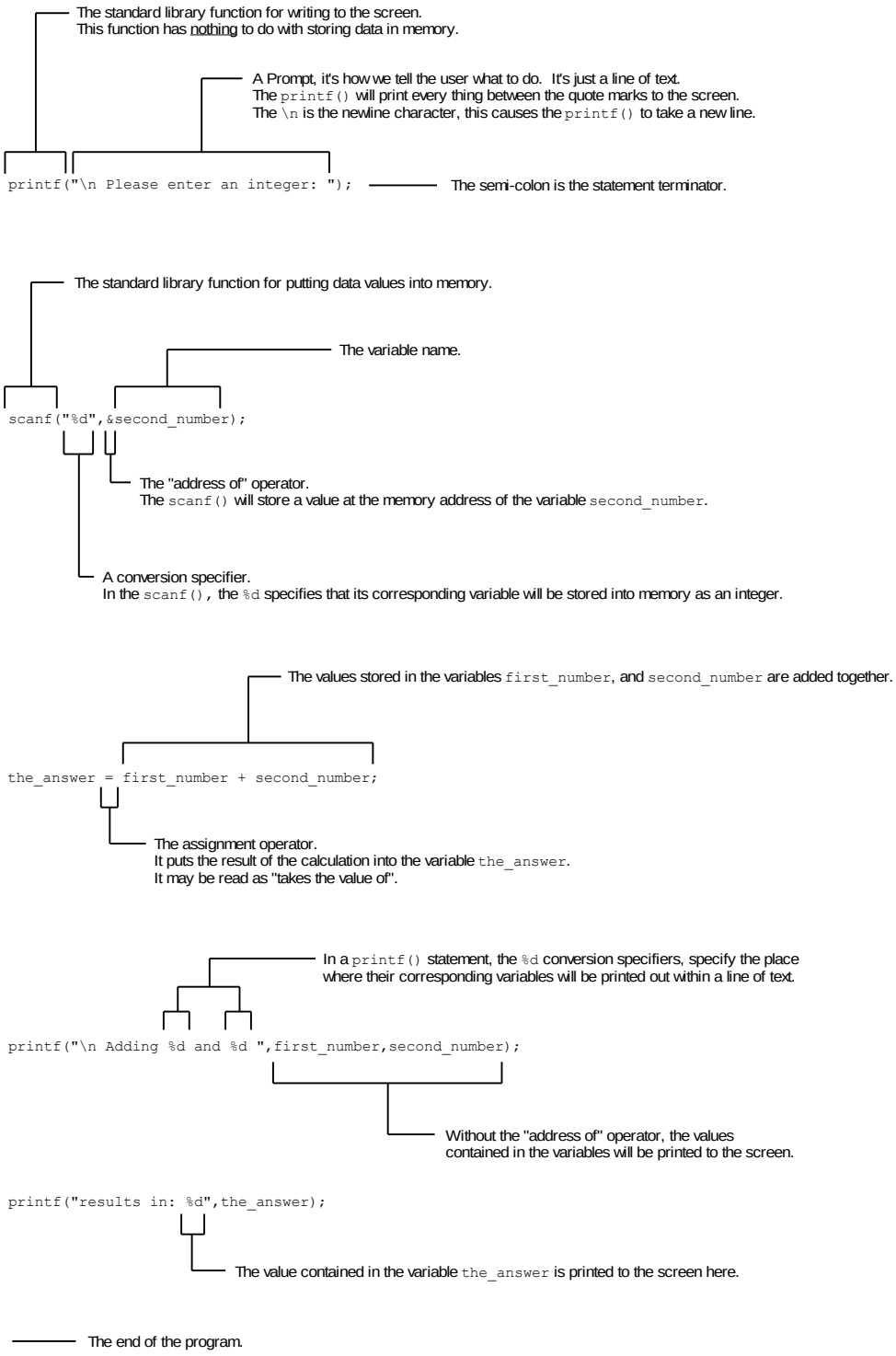
The standard function for putting data values into memory.

The variable name.

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

The "address of" operator.
The scanf() will store a value at the memory address of the variable first_number.

A conversion specifier.
In the scanf(), the %d specifies that its corresponding variable will be stored in memory as an integer.


The standard library function for writing to the screen.
This function has nothing to do with storing data in memory.
A Prompt, it's how we tell the user what to do. It's just a line of text.
The printf() will print every thing between the quote marks to the screen.
The \n is the newline character, this causes the printf() to take a new line.
printf("\n Please enter an integer: ");
The semi-colon is the statement terminator.
The standard library function for putting data values into memory.
The variable name.
scanf("%d",&second_number);
The "address of" operator.
The scanf() will store a value at the memory address of the variable second_number.
A conversion specifier.
In the scanf(), the %d specifies that its corresponding variable will be stored into memory as an integer.
The values stored in the variables first_number, and second_number are added together.
the_answer = first_number + second_number;
The assignment operator.
It puts the result of the calculation into the variable the_answer.
It may be read as "takes the value of".
In a printf() statement, the %d conversion specifiers, specify the place
where their corresponding variables will be printed out within a line of text.
printf("\n Adding %d and %d ",first_number,second_number);
Without the "address of" operator, the values
contained in the variables will be printed to the screen.
printf("results in: %d",the_answer);
The value contained in the variable the_answer is printed to the screen here.
}
The end of the program.

Using the modulus (%) operator

The modulus is the fancy mathematicians' term for what the rest of us call the remainder.

If you divided the whole number 8, by the whole number 3, you would get the result: 2 remainder 1.

In C, the calculation has to be done in two parts, first: 8 / 3, then 8 % 3. There are many uses for this operator, and one of them is demonstrated below in the program `clock.c`

Program (`clock.c`)

```
/*
  Program "clock.c"
  Written by: Joe Dorward
  Date: 03/12/00

  This program demonstrates the use of the modulus (remainder)
  operator %
*/

#include <stdio.h>

void main(void)
{
int minutes_in = 0,
    the_minutes = 0,
    the_hours =0;

  printf("\n Enter the total number of minutes: ");
  scanf("%d",&minutes_in);

  the_hours = minutes_in / 60;
  the_minutes = minutes_in % 60;

  printf("\n The hours are: %d",the_hours);
  printf("\n The minutes are: %d \n",the_minutes);
}
```

If you enter 60 when prompted for the minutes, the output will be:

```
The hours are: 1
The minutes are 0
```

If you enter 90 when prompted for the minutes, the output will be:

```
The hours are: 1
The minutes are 30
```

Formatting number output with field width specifiers

Program (`format1.c`)

```
/*
  Program "format1.c"
  Written by: Joe Dorward
  Date: 03/12/00

  This program demonstrates un-formatted output
*/

#include <stdio.h>

void main(void)
{
  printf("\n Print the integers\n");

  printf("\n %d ",31);
  printf("\n %d ",4);
  printf("\n %d ",321);
  printf("\n %d \n",6);

  printf("\n Now the reals\n");

  printf("\n %f ",78.30);
  printf("\n %f ",4.99);
  printf("\n %f ",126.49);
  printf("\n %f \n",2200.59);

}
```

The program output

What you get

```
Print the integers

 31
 4
 321
 6

 Now the reals

 78.300000
 4.990000
 126.490000
 2200.590000
```

What you want

```
Print the integers

  31
   4
 321
   6

 Now the reals

  78.30
   4.99
 126.49
2200.59
```

To get the output you want, you have to change the program to the version on the next page.

Program (format2.c)

```
/*
  Program "format2.c"
  Written by: Joe Dorward
  Date: 03/12/00

  This program demonstrates formatted output using
  field width specifiers
*/

#include <stdio.h>

void main(void)
{
  printf("\n Print the integers\n");

  printf("\n %4d ",31);
  printf("\n %4d ",4);
  printf("\n %4d ",321);
  printf("\n %4d \n",6);

  printf("\n Now the reals\n");

  printf("\n %7.2f ",78.30);
  printf("\n %7.2f ",4.99);
  printf("\n %7.2f ",126.49);
  printf("\n %7.2f \n",2200.59);

}
```

With the whole numbers, the 4 means the total number of spaces, right aligned.

With the real numbers, the 7 means the total number of spaces, the 2 is the number of spaces after the decimal point.

If you change the numbers, the output will change. Try it.

Program (gas.c)

This next program demonstrates the use of the C keyword `const` which fixes the value contained in the identifier `price_per_gallon` to 1.59.

It also demonstrates the initialization of a float variable, which needs the `'f'` added after the value `1.59`.

```
/*
  Program "gas.c"

  This program asks the user how many gallons of gasoline they want.
  It then calculates the total price, and to the screen.
*/

#include <stdio.h>

void main(void)
{
const float price_per_gallon = 1.59f;   // must add the 'f' when
initializing

float number_of_gallons,
      total_price;

  printf("How many gallons do you want: ");
  scanf("%f",&number_of_gallons);

  total_price = number_of_gallons * price_per_gallon;

  printf("%4.2f gallons of gasoline ",number_of_gallons);
  printf("will cost: $ %4.2f \n",total_price);
}
```

Program (ascii1.c)

```
/*
  Program "ascii1.c"

  This program asks the user how to type in an ASCII character.
  It then prints the character's ASCII value to the screen.
*/

#include <stdio.h>

void main(void)
{
char the_character;

  printf(" Please enter the character: ");
  scanf("%c",&the_character);

  printf("\n The ASCII value of: %c, ",the_character);
  printf("is: %d \n",the_character);
}
```