COE 251

DATA INPUT AND OUTPUT

Dr. Eliel Keelson

OUTLINE

COMMENTS

MY FIRST PROGRAM

DATA INPUT/OUTPUT

INTENDED LEARNING OUTCOMES (ILOs)

To be able to use and identify Comments

To Understand how to Output Data on a Display

To Understand how to Input Data via the Keyboard

1

COMMENTS

COMMENTS

- A **comment** is a note to yourself (or others) that you put into your source code.
- All comments are ignored by the compiler. They exist solely for your benefit.
- Comments are used primarily to document the meaning and purpose of your source code, so that you can remember later how it functions and how to use it.

COMMENTS

- Comments are however not mandatory. A developer may decide not to include them in code.
- However, it is good programming practice to include comments.
- In C, the start of a comment is signaled by the **/*** character pair. A comment is ended by ***/**
- For example, this is a syntactically correct C comment: **/* This is a comment. */**

COMMENTS

- Comments that span over multiple lines can be encapsulated just as shown earlier.
- However, single line comments can be started with **//** such as **//This is a single line comment**

> “*Commenting your code is like cleaning your bathroom - you never want to do it, but it really does create a more pleasant experience for you and your guests.* — Ryan Campbell

2

FIRST PROGRAM

Understanding and Running Your First Program in C

MY FIRST PROGRAM – HELLO WORLD

- The Code on the next slide is an example of a simple program to get you started in C.

MY FIRST PROGRAM – HELLO WORLD

```
/*
Finally My Very
first C program
*/

#include <stdio.h>
int main()
{
    printf("Hello, World!\n");   // displays Hello, World! on the screen
    return 0;
}
```

MY FIRST PROGRAM – HELLO WORLD

- In the code above it is very obvious that some of the text in there are just comments.
- Can you identify any multi-line and/or single line comments?

#include <stdio.h>

- The program seen above displays to the screen the text “**Hello World!**”
- To do this successfully the program needs to speak to the computer’s hardware responsible for display.
- This requires an elaborate sequence of actions. However, this heavy lifting has been done for us and bundled in a **Header File** called **stdio** (Standard Input/Output) header file.

#include <stdio.h>

- The **.h** simply connotes that this is a header file.
- This header file contains a library of functions responsible for input and output operations, such as displaying to the computer screen using the **printf** (print function).
- So to use any of these functions one must first include it into the code using the **#include** preprocessor command.
- A lot more would be said on preprocessor commands later.

The main() Function

- In C programming, the code execution begins from the start of **main()** function (doesn’t matter if **main()** isn’t located at the beginning of the code).
- The code inside the curly braces **{ }** is the **body** of **main()** function.
- The **main()** function is mandatory in every C program.

The main() Function

- The data type **int** that comes before the **main()** function is simply to cater for the **returning** of the integer **0** from the statement **return 0**.
- A lot more would be shared on the returning value when studying the topic **FUNCTIONS.**

The printf() Function

- The **printf()** is a library function that sends formatted output to the screen (displays the string inside the quotation marks).
- Notice the semicolon(**;**) at the end of the statement.
- In our program, it displays **Hello, World!** on the screen.
- Remember, you need to include **stdio.h** file in your program for this to work.

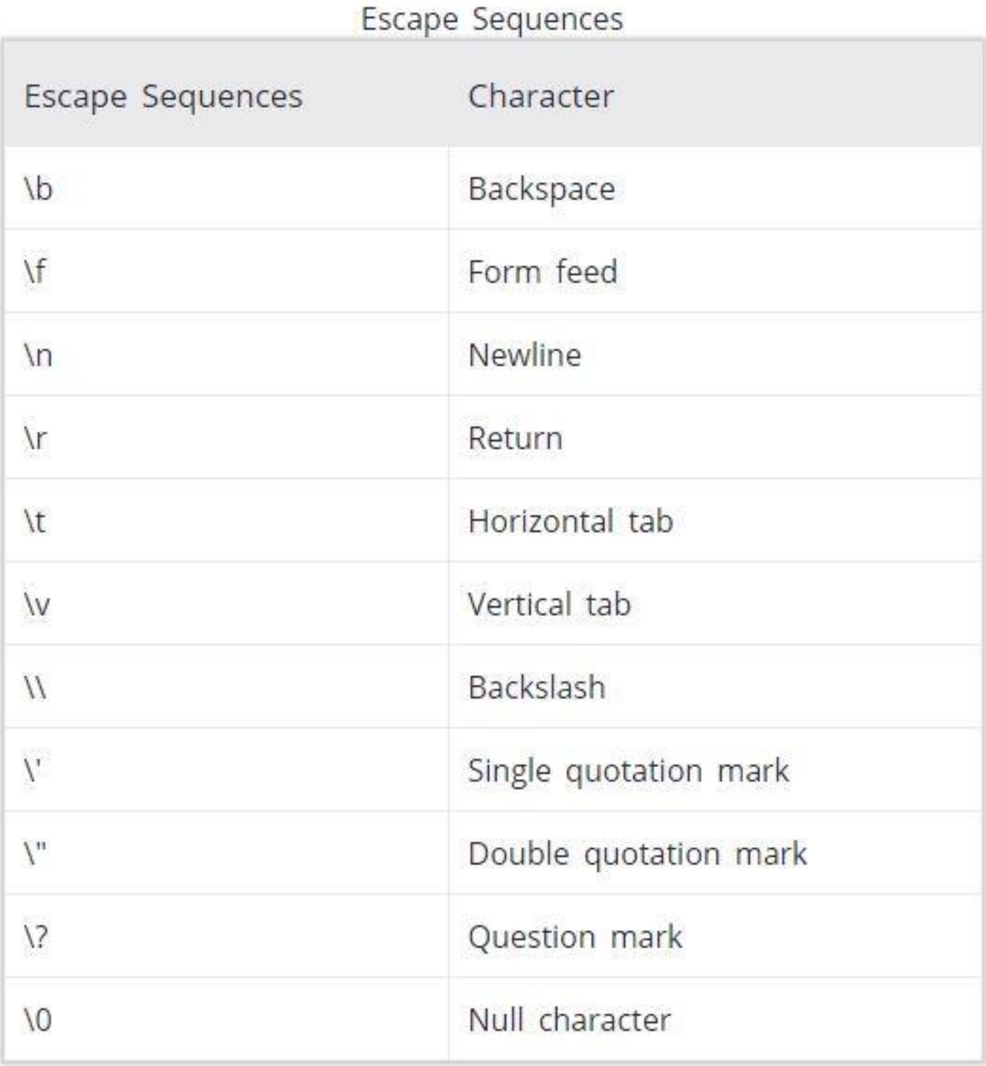
The return Statement

- The return statement **return 0;** inside the **main()** function ends the program.
- This statement isn’t mandatory. However, it’s considered good programming practice to use it.

ESCAPE SEQUENCES

- You must have noticed a **\n** pair of characters inside the **printf()** function. This pair of characters is an example of an **Escape Sequence.**
- The combination of these characters provide a special effect. For example the **\n** produces a newline (like the enter key) and the **\t** produces a horizontal space (like the tab key)

ESCAPE SEQUENCES

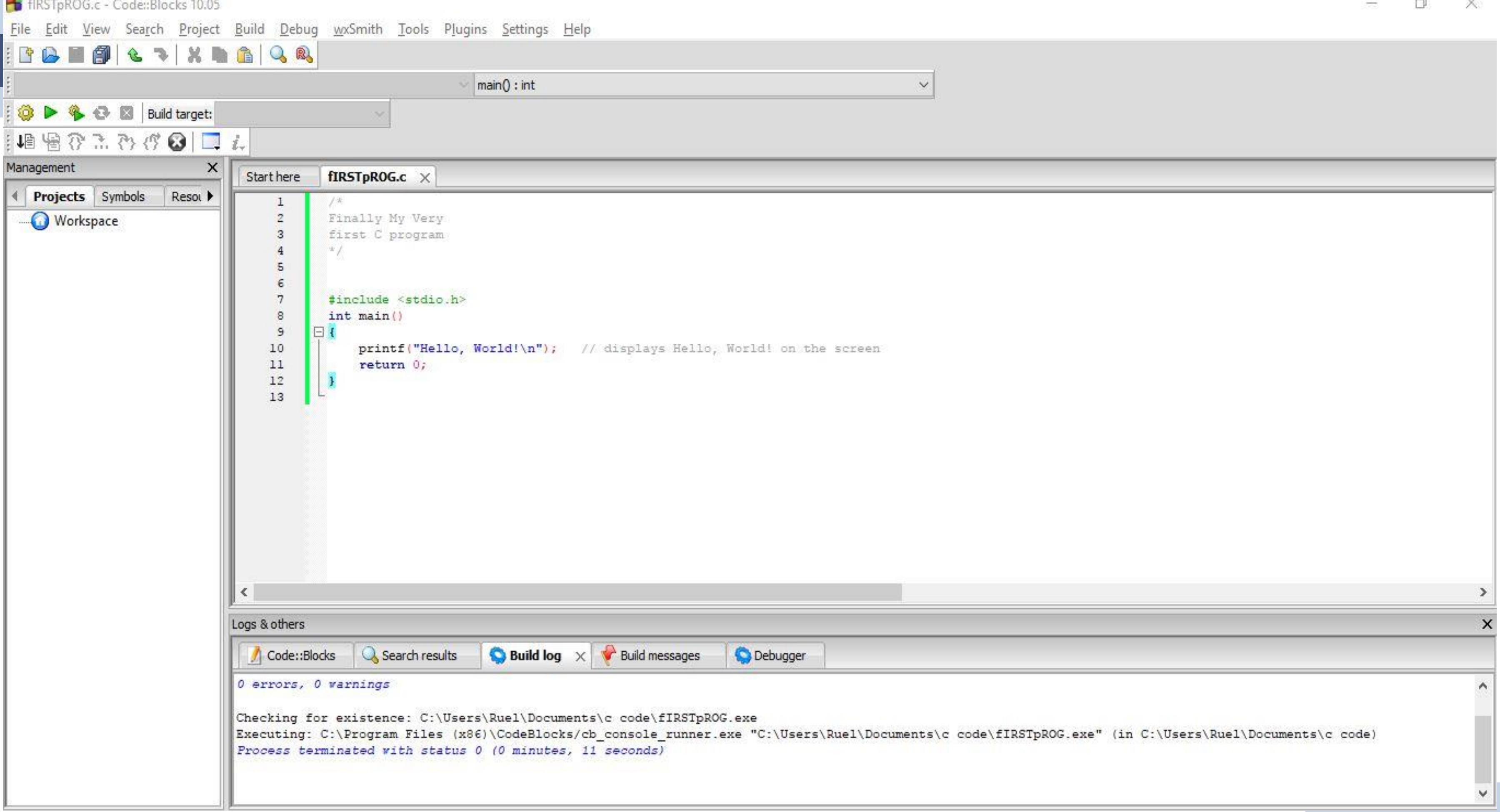


Escape Sequences

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

RUNNING THE CODE

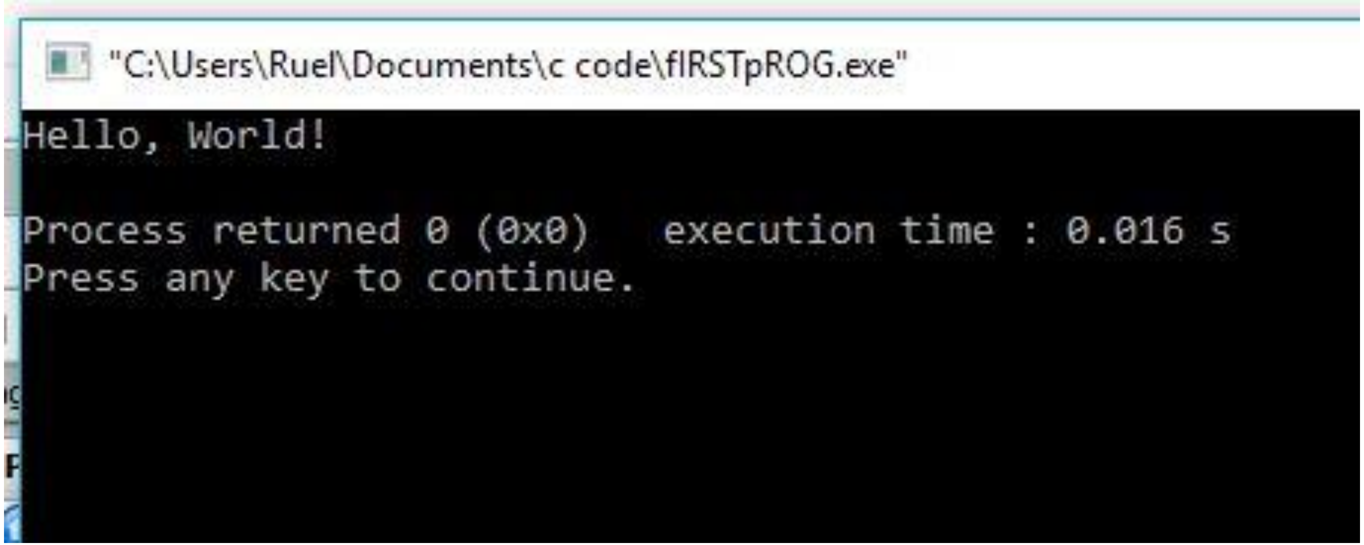
- The code presented above would be typed into a favourable Integrated Development Environment (IDE) that has support for C, for example CodeBlocks.
- After saving the code with a file name having an extension **.c** the code would be built and ran/executed.
- After running the code the result would be displayed on the screen.



```
/*
Finally My Very
first C program
*/


#include <stdio.h>
int main()
{
    printf("Hello, World!\n");    // displays Hello, World! on the screen
    return 0;
}

```

```
0 errors, 0 warnings

Checking for existence: C:\Users\Ruel\Documents\c code\fIRSTpROG.exe
Executing: C:\Program Files (x86)\CodeBlocks/cb_console_runner.exe "C:\Users\Ruel\Documents\c code\fIRSTpROG.exe" (in C:\Users\Ruel\Documents\c code)
Process terminated with status 0 (0 minutes, 11 seconds)
```

RUNNING THE CODE



"C:\Users\Ruel\Documents\c code\fIRSTpROG.exe"

```
Hello, World!

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

3

DATA INPUT AND OUTPUT

DATA INPUT/OUTPUT

- From the very first program, to some extent, it is understood by what data output refers to; displaying data to the user.
- The reverse; getting data from the user, can also be described as data input.
- With the help of functions in the **stdio.h** file, these operations are made possible.

DATA INPUT/OUTPUT

- There are six main functions in the stdio.h that would be of immense use in this course.
- They are: **printf()**, **scanf()**, **putchar()**, **getchar()**, **puts()** and **gets()**
- The general syntax and mode of usage of each of these would be explained in the following slides.

PRINTF

- The **printf()** function, as seen already, writes output data from the computer to a standard output device.
- This function can be used to output any combination of numerical values, single characters and strings.
- Strings are simply a concatenation of characters.

PRINTF

- The general syntax for using **printf** is :

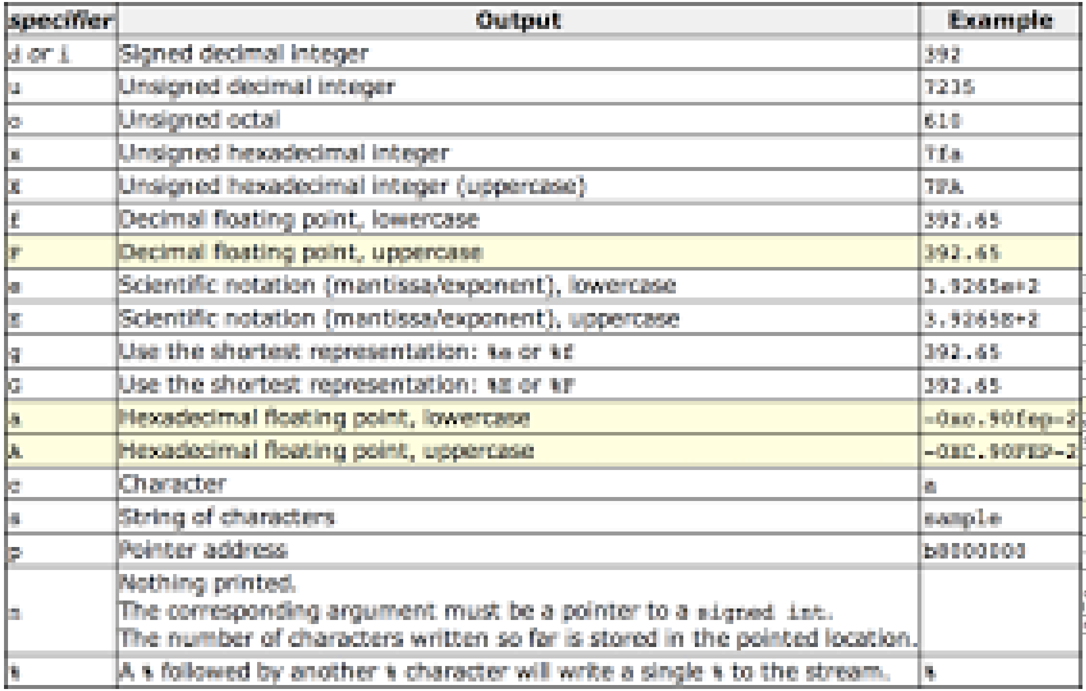
  printf("control string", arg1, arg2,…, argn);

- where **control string** refers to a string containing certain formatting information, and **arg_1**, **arg_2**,…, **arg_n** are arguments that represent the individual output data items.

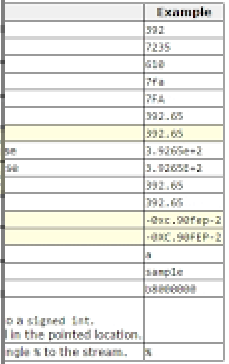
PRINTF

- Control strings are also known as string modifiers/formatters/specifiers.
- While using **printf()**, these string formatters have the ability to specify how values from variables are displayed.
- The next slide presents some of these control strings.

CONTROL STRINGS



specifier	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%



	Example
	392
	7235
	610
	7fa
	7FA
	392.65
	392.65
e	3.9265e+2
se	3.9265E+2
	392.65
	392.65
	-0xc.90fep-2
	-0XC.90FEP-2
	a
	sample
	b8000000
o a signed int. l in the pointed location.	
ngle % to the stream.	%

PRINTF

- However, printf() statements do not always contain control strings.
- For example, what was written in the first program.

```
printf(“Hello, World!\n”);
printf(“I am 50 years old”);
```

PRINTF

- The above examples would print exactly what has been capsuled in the quotation marks (together with the effect of any escape sequences).
- However, there are times we would want to print values that are stored in variables (memory).
- To do so we would have to use control strings.

PRINTF

- For example:

```
int age = 50; //the variable is age and the value stored is 50
printf(“%d \n”, age); // to print the value 50 as an integer
Printf(“I am %d years old”, age); // printing the age from memory
```

PRINTF

- For example:

```
int score = 4; //the variable is score and the value stored is 4
float weight = 90.61;
printf(“Zlatan Ibrahimovic who scored %d goals in the last game weighs %g kilograms”, score, weight);
// combining various various variables in one print statement
```

PRINTF

- In the last example, a number of variables were combined in one **printf()** statement.
- It is important to take close notice of how the control strings and their respective variables are ordered.
- Misplacing them would not produce the desired output.

PRINTF

- The control strings can also be used to specify the length of value to be displayed as well as how many decimal points a floating point has.
- Find out more on how these are done.

SCANF

- Input data can be entered into the computer from a standard input device by means of a C library function by name **scanf()**.
- This function can be used to input any combination of numerical values, single characters and strings.

SCANF

- The general syntax for using **scanf** is:

  scanf(“control string”, arg_1, arg_2,…, arg_n);

- where **control string** refers to a string containing certain formatting information, and **arg_1**, **arg_2**,…, **arg_n** are arguments that represent the individual input data items.

SCANF

- A lot has already been said on control strings. So it’s okay to start taking examples.
- Imagine if you were to take the age from a user. It can be done doing the following.

```
int personAge;
scanf(“%d”, &personAge);
```

SCANF

- We first declared the integer variable **personAge** which would hold the person’s age.
- Then using the **scanf()** statement we patiently wait for the person to enter the age and store the age in the declared variable called **personAge**.
- So when the compiler gets to the line with the **scanf()** statement it would keep blinking a cursor until the person enters a value

SCANF

- The entered value would be treated as an integer because we specified **%d** as the control string.
- The entered value is stored in the variable **personAge**.
- Once stored it can always be retrieved later in the program.

& - THE "ADDRESS OF" OPERATOR

- You must have been wondering why we placed an ampersand (**&**) before the variable **personAge** when using the **scanf** statement.
- This ampersand is known as the "address of" operator and it is used for accessing the address (memory location) of the declared variable where we want to store the value.

& - THE “ADDRESS OF” OPERATOR

- Without the ampersand (**&**) the **scanf** statement would not work as we want it to.
- The statement **&personAge** would provide the compiler with the address of the variable **personAge**.
- This address of operator can also be used in the **printf** statement to print out the memory addresses of variables using the control string **%p**. More of this would be covered later when treating the topic **Pointers**.

SCANF – MULTIPLE USER INPUT

- There are times when there would be the need of taking multiple user inputs.
- For example taking 3 exam scores from the user in order to compute the average score.
- There are two ways of doing this. The next few slides would demonstrate this.

SCANF – MULTIPLE USER INPUT

One way of getting it done is by writing the **scanf** statement each time a value is to be taken. For example:

```
float score_1, score_2, score_3;
scanf(“%g”, &score_1);
scanf(“%g”, &score_2);
scanf(“%g”, &score_3);
```

SCANF – MULTIPLE USER INPUT

Another way of getting it done is by writing the **scanf** statement just once, while comma-separating the arguments. For example:

```
float score_1, score_2, score_3;
scanf("%g %g %g", &score_1, &score_2, &score_3);
```

It should however be noted that the control strings are not comma-separated.

GETCHAR AND PUTCHAR

- **getchar()** and **putchar()** are also functions in the standard input and output header file which work mainly with characters.
- **getchar()** is responsible for taking a character value from a user (input) and **putchar()** is responsible for displaying a character value to a user.

GETCHAR AND PUTCHAR

- The syntax for **getchar()** is

declared character variable = getchar();

- For example

char choice = getchar();

- Since it is an input function, this would cause the compiler to patiently wait while blinking a cursor until the user enters a character value which would be stored in the variable **choice**.

GETCHAR AND PUTCHAR

- The syntax for **putchar()** is

putchar(character variable);

- For example

putchar(choice);

- Since it is an output function, this would cause the compiler to display the character value which was stored in the variable **choice**.

GETS AND PUTS

- The **gets()** and **puts()** functions are also input and output functions targeted mainly for string values.
- How they are used would be covered later when treating the topic **Strings**.

THANKS!

Any questions?

You can find me at

elielkeelson@gmail.com & ekeelson@knust.edu.gh