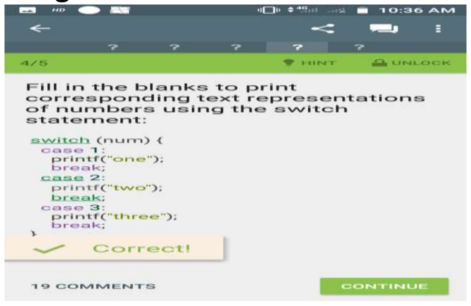
DAILY ASSESSMENT FORMAT

Date:	**22-06-2020**	**Name:**	**PRINCIA MELITA DSOUZA**
Course:	**sololearn**	**USN:**	**4AL17EC075**
Topic:	**C++**	**Semester & Section:**	**6TH B**
Github Repository:	**MELITA-1999**		

FORENOON SESSION DETAILS

Image of session



Fill in the blanks to print corresponding text representations of numbers using the switch statement:

```
switch (num) {
  case 1:
    printf("one");
    break;
  case 2:
    printf("two");
    break;
  case 3:
    printf("three");
    break;
}
```

Report – Report can be typed or hand written for up to two pages.

Error Handling

Suppose we are dividing two numbers and one of them is zero. In this case the program will not automatically handle the error of dividing by zero. The user needs to explicitly check for the numbers and display proper message. We need to send -1 or NULL value to the calling program or operating system. C does not provide any exception handling or error handling features like any other programming language. But C allows error handling by using the variables and functions defined in the header file "errno.h". It has error code in a variable 'errno' – a global variable and contains different error codes for different errors. Hence at the beginning of the program, errno is initialized to zero to indicate that it does not reflect any unwanted errors.The last chapter explained the standard input and output devices handled by C programming language. This chapter cover how C programmers can create, open, close text or binary files for their data storage.A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices. This chapter will take you through the important calls for file management.

Opening Files

You can use the **fopen( )** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. The prototype of this function call is as follows −

FILE *fopen(const char * filename, const char * mode);

Here, **filename** is a string literal, which you will use to name your file, and access **mode** can have one of the following values −

Sr.No.	Mode & Description
1	**r** Opens an existing text file for reading purpose.
2	**w** Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
3	**a** Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
4	**r+** Opens a text file for both reading and writing.
5	**w+** Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
6	**a+** Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

If you are going to handle binary files, then you will use following access modes instead of the above mentioned ones −

"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

Closing a File

To close a file, use the fclose( ) function. The prototype of this function is −

```
int fclose( FILE *fp );
```

The **fclose(-)** function returns zero on success, or **EOF** if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file **stdio.h**.

There are various functions provided by C standard library to read and write a file, character by character, or in the form of a fixed length string.

Writing a File

Following is the simplest function to write individual characters to a stream −

int fputc(int c, FILE *fp);

The function **fputc()** writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success otherwise **EOF** if there is an error. You can use the following functions to write a null-terminated string to a stream −

int fputs(const char *s, FILE *fp);

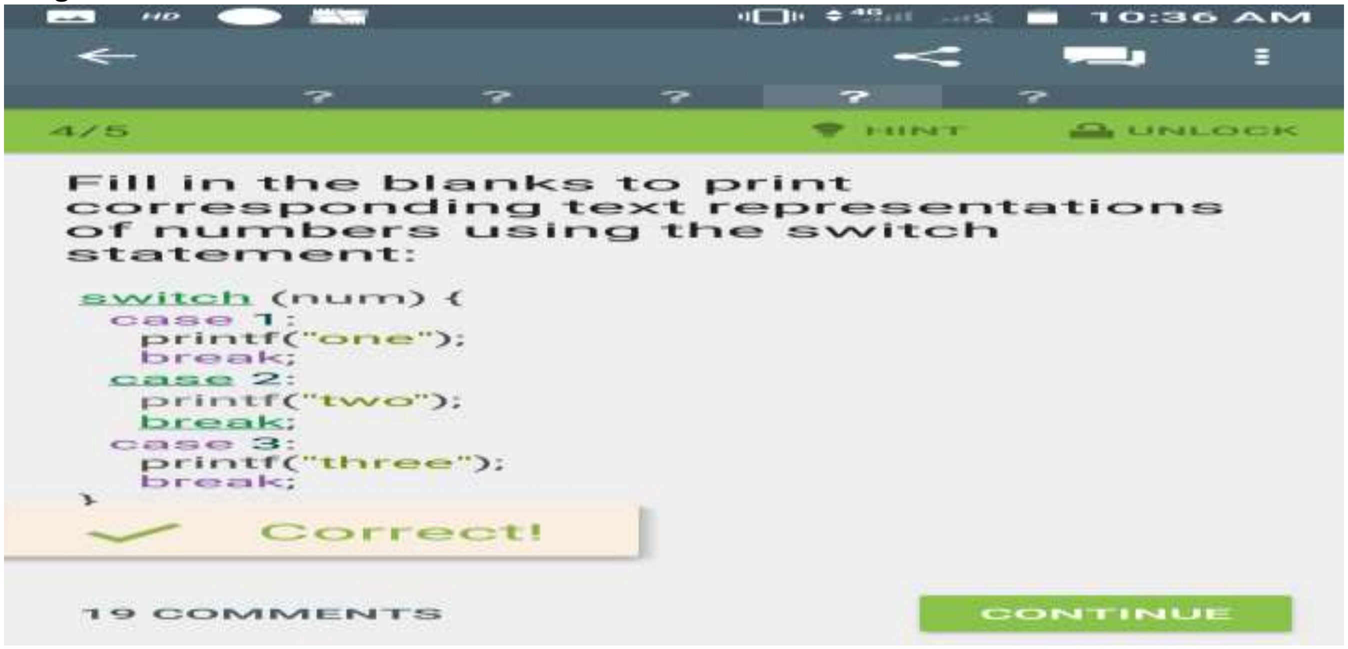
The function **fputs()** writes the string **s** to the output stream referenced by fp. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error. You can use **int fprintf(FILE *fp,const char *format, ...)** function as well to write a string into a file. Try the following example.

Make sure you have **/tmp** directory available. If it is not, then before proceeding, you must create this directory on your machine.

Date:	22-06-2020	Name:	PRINCIA
Course:	sololearn	USN:	4AL17EC075
Topic:	C++	Semester & Section:	6th B

AFTERNOON SESSION DETAILS

Image of session



Fill in the blanks to print corresponding text representations of numbers using the switch statement:

```
switch (num) {
  case 1:
    printf("one");
    break;
  case 2:
    printf("two");
    break;
  case 3:
    printf("three");
    break;
}
```

Report – Report can be typed or hand written for up to two pages.

The C Preprocessor

The C preprocessor is a *macro processor* that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define *macros*, which are brief abbreviations for longer constructs.

The C preprocessor provides four separate facilities that you can use as you see fit:

- Inclusion of header files. These are files of declarations that can be substituted into your program.
- Macro expansion. You can define *macros*, which are abbreviations for arbitrary fragments of C code, and then the C preprocessor will replace the macros with their definitions throughout the program.
- Conditional compilation. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.
- Line control. If you use a program to combine or rearrange source files into an intermediate file which is then compiled, you can use line control to inform the compiler of where each source line originally came from.

C preprocessors vary in some details. This manual discusses the GNU C preprocessor, the C Compatible Compiler Preprocessor. The GNU C preprocessor provides a superset of the features of ANSI Standard C.ANSI Standard C requires the rejection of many harmless constructs commonly used by today's C programs. Such incompatibility would be inconvenient for users, so the GNU C preprocessor is configured to accept these constructs by default. Strictly speaking, to get ANSI Standard C, you must use the options `-trigraphs', `-undef' and `-pedantic', but in practice the consequences of having strict ANSI Standard C make it undesirable to do this.The C preprocessor is designed for C-like languages; you may run into problems if you apply it to other kinds of languages, because it assumes that it is dealing with C. For example, the C preprocessor sometimes outputs extra white space to avoid inadvertent C token concatenation, and this may cause problems with other languages.