**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **20/6/2020** | **Name:** | **M V Ramya** |
| **Course:** | |  | | --- | | **C programming** | | **USN:** | **4AL17EC045** |
| **Topic:** | * **File and Error handling** * **The Preprocessor** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **MV-Ramya-045** |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **FORENOON SESSION DETAILS** | | | |
| **Image of the session** | | | |
| **REPORT-**  **Report**  **Files & Error Handling**   * **Accessing Files** - An external file can be opened, read from, and written to in a C program. For these operations, C includes the FILE type for defining a file stream. The file stream keeps track of where reading and writing last occurred. * The **stdio.h** library includes file handling functions: * FILE **Typedef** for defining a file pointer. * **fopen**(filename, mode) Returns a FILE pointer to file filename which is opened using mode. If a file cannot be opened, NULL is returned. * Mode options are: * r open for reading (file must exist). * w open for writing (file need not exist). * a open for append (file need not exist). * r+ open for reading and writing from beginning. * w+ open for reading and writing, overwriting file. * a+ open for reading and writing, appending to file. * **fclose**(fp) Closes file opened with FILE fp, returning 0 if close was successful. EOF (end of file) is returned if there is an error in closing. * **Reading from a File** - The stdio.h library also includes functions for reading from an open file. A file can be read one character at a time or an entire string can be read into a character buffer, which is typically a char array used for temporary storage. * **fgetc**(fp) Returns the next character from the file pointed to by fp. If the end of the file has been reached, then **EOF** is returned. * **fgets**(buff, n, fp) Reads n-1 characters from the file pointed to by fp and stores the string in buff. A NULL character '\0' is appended as the last character in buff. If fgets encounters a newline character or the end of file before n-1 characters is reached, then only the characters up to that point are stored in buff. * **fscanf**(fp, conversion\_specifiers, vars) Reads characters from the file pointed to by fp and assigns input to a list of variable pointers vars using conversion\_specifiers. As with scanf, fscanf stops reading a string when a space or newline is encountered. * **Writing to a File** - The stdio.h library also includes functions for writing to a file. When writing to a file, newline characters '\n' must be explicitly added. * **fputc**(char, fp) Writes character char to the file pointed to by fp. * **fputs**(str, fp) Writes string str to the file pointed to by fp. * **fprintf**(fp, str, vars) Prints string str to the file pointed to by fp. str can optionally include format specifiers and a list of variables vars. * **Exception Handling** - Central to good programming practices is using error handling techniques. Even the most solid coding skills may not keep a program from crashing should you forget to include exception handling. An exception is any situation that causes your program to stop normal execution. Exception handling, also called error handling, is an approach to processing runtime errors.      * Some library functions, such as **fopen()**, set an error code when they do not execute as expected. The error code is set in a global variable named **errno**, which is defined in the **errno.h** header file. When using **errno** you should set it to 0 before calling a library function. * To output the error code stored in **errno**, you use fprintf to print to the stderr file stream, the standard error output to the screen. Using **stderr** is a matter of convention and a good programming practice. * You can output the **errno** through other means, but it will be easier to keep track of your exception handling if you only use **stderr** for error messages. * When a library function sets **errno**, a cryptic error number is assigned. For a more descriptive message about the error, you can use **perror()**. You can also obtain the message using **strerror()** in the string.h header file, which returns a pointer to the message text.      * Some of the mathematical functions in the **math.h** library set errno to the defined macro value **EDOM** when a domain is out of range. Similarly, the **ERANGE** macro value is used when there is a range error. * In addition to checking for a NULL file pointer and using **errno**, the **feof()** and **ferror()** functions can be used for determining file I/O errors: * **feof**(fp) Returns a nonzero value if the end of stream has been reached, 0 otherwise. feof also sets EOF. * **ferror**(fp) Returns a nonzero value if there is an error, 0 for no error.   **The Preprocessor**  The **C preprocessor** uses the **# directives** to make substitutions in program source code before compilation.   * For example, the line **#include <stdio.h>** is replaced by the contents of the **stdio.h** header file before a program is compiled. * **Preprocessor directives** and their uses: * **#include** Including header files. * **#define**, #undef Defining and undefining macros. * **#ifdef**, **#ifndef**, **#if**, **#else**, **#elif**, **#endif** Conditional compilation. * **#pragma** Implementation and compiler specific. * **#error**, **#warning** Output an error or warning message An error halts compilation. * **Formatting Preprocessor Directives** - When using preprocessor directives, the # must be the first character on a line. But there can be any amount of white space before # and between the # and the directive. If a # directive is lengthy, you can use the \ continuation character to extend the definition over more than one line. * In addition to defining your own macros, there are several standard predefined macros that are always available in a C program without requiring the **#define** directive: * **\_\_DATE\_\_:** The current date as a string in the format Mm dd yyyy. * **\_\_TIME\_\_:** The current time as a string in the format hh:mm:ss. * **\_\_FILE\_\_:** The current filename as a string. * **\_\_LINE\_\_:** The current line number as an int value. * \_\_STDC\_\_: 1. * The **# macro operator** is called the stringification or stringizing operator and tells the preprocessor to convert a parameter to a string constant. White space on either side of the argument are ignored and escape sequences are recognized. * The **## operator** is also called the token pasting operator because it appends, or "pastes", tokens together. | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  | | | |