**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **20/06/2020** | **Name:** | **Lepakshi T V** |
| **Course:** | **C programming** | **USN:** | **4AL17EC044** |
| **Topic:** | * **File and error handling** * **The preprocessors** | **Semester & Section:** | **6th sem A sec** |
| **Github Repository:** | **Lepakshi-044** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report – Report can be typed or hand written for up to two pages.**  **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. C does not explicitly support exception handling, but there are ways to manage errors: - Write code to prevent the errors in the first place. You can't control user input, but you can check to be sure that the user entered valid input. When performing division, take the extra step to ensure that **division by 0** won't occur. - Use the **exit** statement to gracefully end program execution. You may not be able to control if a file is available for reading, but you don't need to allow the problem to crash your program.  **Preprocessor Directives** 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. Preprocessor Operators The C preprocessor provides the following operators.The # Operator 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** The **## operator** is also called the **token pasting** operator because it appends, or "pastes", tokens together.  The program below demonstrates these directives:  #include <stdio.h>  #define RATE 0.08 **#ifndef TERM** #define TERM 24 **#endif**  int main() { **#ifdef** RATE /\* this branch will be compiled \*/ **#undef** RATE  printf("Redefining RATE\n"); #define RATE 0.068 **#else** /\* this branch will not be compiled \*/ #define RATE 0.068 **#endif**  printf("%f %d\n", RATE, TERM);  return 0; } |