**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **20/06/2020** | **Name:** | **Kirti B S** |
| **Course:** | **C Programming** | **USN:** | **4AL18EC026** |
| **Topic:** | * **Files and Error Handling** * **The Preprocessor** | **Semester & Section:** | **4th sem ‘A’ section.** |
| **Github Repository:** | **Kirti BS** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**  **C:\Users\hello\Pictures\Screenshots\Screenshot (286).png**  **C:\Users\hello\Pictures\Screenshots\Screenshot (287).png** |
| **REPORT**  **In today’s session I have learnt about:**  **Chapter 1: Files and Error Handling**   * **Working with Files:**  Accessing Files  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 openedusing *mode*. If a file cannot be opened, NULL is returned.  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.   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*.** 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.** Binary File I/OControlling the File Pointer **There are functions in stdio.h for controlling the location of the file pointer in a binary file: ftell(fp) Returns a long int value corresponding to the *fp* file pointer position in number of bytes from the start of the file.**   * **Error handling**  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.The exit Command **The exit command immediately stops the execution of a program and sends an exit code back to the calling process. For example, if a program is called by another program, then the calling program may need to know the exit status.**   * **Using Error Codes**  Using errno **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.  To use errno, you need to declare it with the statement extern int errno; at the top of your program (or you can include the errno.h header file).** The perror and strerror Functions **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. perror() must include a string that will precede the actual error message.** EDOM and ERANGE Error Codes **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.** The feof and ferror Functions **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.**  **Chapter 2: The Preprocessor**   * **Preprocessor Directives**  Preprocessor Directives **The C preprocessor uses the # directives to make substitutions in program source code before compilation. 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.** The #include Directive **The #include directive is for including header files in a program. A header file declares a collection of functions and macros for a library, a term that comes from the way the collection of code can be reused.  Corresponding header files for the libraries end with .h by convention. The #include directive expects brackets <> around the header filename if the file should be searched for in the compiler include paths.  A user-defined header file is also given the .h extension, but is referred to with quotation marks, as in "myutil.h". When quotation marks are used, the file is searched for in the source code directory.** The #define Directive **The #define directive is used to create object-like macros for constants based on values or expressions. #define can also be used to create function-like macros with arguments that will be replaced by the preprocessor.** 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.** Predefined Macro Definitions **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**   * **Conditional Compilation Directives**  The #ifdef, #ifndef, and #undef Directives **The #ifdef, #ifndef, and #undef directives operate on macros created with #define. For example, there will be compilation problems if the same macro is defined twice, so you can check for this with an #ifdef directive. Or if you may want to redefine a macro, you first use #undef.** Conditional Compilation Directives **Conditional compilation of segments of code is controlled by a set of directives: #if, #else, #elif, and #endif.**   * **Preprocessor operators**  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.** |