**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **18/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **C Programming** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Module 7: Files & Error Handling** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **SESSION DETAILS** |
|  |

|  |
| --- |
| **Report** 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. The following program opens a file for writing and then closes it: #include <stdio.h> int main() {  FILE \*fptr; fptr = fopen("myfile.txt", "w"); if (fptr == NULL) { printf("Error opening file."); return -1; } fclose(fptr); return 0; } 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. The following program demonstrates reading from a file: #include <stdio.h> int main() {  FILE \*fptr; int c, stock; char buffer[200], item[10]; float price; /\* myfile.txt: Inventory\n100 Widget 0.29\nEnd of List \*/ fptr = fopen("myfile.txt", "r"); fgets(buffer, 20, fptr); /\* read a line \*/ printf("%s\n", buffer); fscanf(fptr, "%d%s%f", &stock, item, &price); /\* read data \*/ printf("%d %s %4.2f\n", stock, item, price); while ((c = getc(fptr)) != EOF) /\* read the rest of the file \*/ printf("%c", c); fclose(fptr); return 0; } 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. |

**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **18/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **C Programming** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Module 8: The Processors** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **SESSION DETAILS** |
|  |

|  |
| --- |
| **Report** 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.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. Some useful C libraries are: stdio input/output functions, including printf and file operations. stdlib memory management and other utilities string functions for handling strings errno errno global variable and error code macros math common mathematical functions time time/date utilities 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 #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.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. For example: #define TO\_STR(x) #x printf("%s\n", TO\_STR( 123\\12 )); Conditional Compilation Directives Conditional compilation of segments of code is controlled by a set of directives: #if, #else, #elif, and #endif. For example: #define LEVEL 4 int main() { #if LEVEL > 6 /\* do something \*/ #elif LEVEL > 5 /\* else if branch \*/ #elif LEVEL > 4 /\* another else if \*/ #else /\* last option here \*/ #endif return 0; } There are instances where such conditional compilation can be useful, but this type of code should be used sparingly. The defined() preprocessor operator can be used with #if, as in:#if !defined(LEVEL) /\* statements \*/ #endif The #if and if statement are not interchangeable. The #if is evaluated using data available to the preprocessor, which then sends only the true branch for compilation.The ## Operator The ## operator is also called the token pasting operator because it appends, or "pastes", tokens together. For example: #define VAR(name, num) name##num int x1 = 125; int x2 = 250; int x3 = 500;  printf("%d\n", VAR(x, 3)); |

