**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **20 June 2020** | **Name:** | **Kavya M M** |
| **Course:** | **C programming** | **USN:** | **4AL17EC040** |
| **Topic:** | **Module 7: Files & Error Handling**  **Module 8: The Processors** | **Semester & Section:** | **6TH SEM & ‘A SEC** |
| **Github Repository:** | **Kavya\_ECE040** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**    **C:\Users\cw\Desktop\20 j2.png** |
| **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.  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  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.  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.  The following program demonstrates writing to a file:  FILE \*fptr;  char filename[50];  printf("Enter the filename of the file to create: ");  gets(filename);  fptr = fopen(filename, "w");  /\* write to file \*/  fprintf(fptr, "Inventory\n");  fprintf(fptr, "%d %s %f\n", 100, "Widget", 0.29);  fputs("End of List", fptr);  **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.  **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.  **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.  FILE \*fptr;  int c;  errno = 0;  fptr = fopen("myfile.txt", "r");  if (fptr == NULL) {  fprintf(stderr, "Error opening file. %s\n", strerror(errno));  exit(EXIT\_FAILURE);  }  while ((c = getc(fptr)) != EOF) /\* read the rest of the file \*/  printf("%c", c);  if (ferror(fptr)) {  printf("I/O error reading file.");  exit(EXIT\_FAILURE);  }  else if (feof(fptr)) {  printf("End of file reached.");  }  **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.  For example:  #include <stdio.h>  #include “myutil.h”  **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.  Be cautious with function-like definitions. Keep in mind that the preprocessor does a direct replacement without any calculations, which can lead to unexpected results, as demonstrated with the following program:  #include <stdio.h>  #define PI 3.14  #define AREA(r) (PI\*r\*r)  int main() {  float radius = 2;  printf("%3.2f\n", PI);  printf("Area is %5.2f\n", AREA(radius));  printf("Area with radius + 1: %5.2f\n", AREA(radius+1));  return 0;  }  **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.  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;  }  **Conditional Compilation Directives**  Conditional compilation of segments of code is controlled by a set of directives: #if, #else, #elif, and #endif. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |