**Data Structures and Algorithms**

Lab 4: File handling

**Q1):** Evaluating execution time:

The library <Time.h> has three important elements:

1. clock\_t: this is our data-type,

2. CLOCKS\_PER\_SEC: a reserved word which evaluatesthe number of clocks the timer makes persecond. This variable changes for different processors. Faster processors have higher clocks per second.

3. clock(): function that measures the number of clocks at a particular instance. It returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by CLOCKS\_PER\_SEC

**Code:**

**#include <iostream>**

**#include <time.h>**

**using namespace std;**

**void fun(){**

**cout<<"Fun() start \n";**

**while (1)**

**{**

**if (getchar())**

**{**

**}**

**}**

**cout<<"Fun() ends \n";**

**}**

**// main function here**

**int main()**

**{**

**clock\_t t1,t2;**

**t1 = clock();**

**fun();**

**t2 = clock() - t1;**

**double time\_taken = ((double) t2) / CLOCKS\_PER\_SEC;**

**cout<<"fun() took of seconds to execute \n"<<time\_taken;**

**return 0;**

**}**

**Output:**



Fig: 01

**Q#02:** Feeding parameters directly from console:

There are two extremely important arguments of the main function, namely, argc and argv[]. Here ‘argc’ stands for argument count and ‘argv’ stands for argument values. These are variables passed to the main function when it starts executing. When we run a program we can give arguments to that program like:

C:\Users\MyComputer\Downloads> Hello.exe 7

Here the argument 7 is directly provided in the console and the program will print Hello a total of seven times. Consider the code mentioned below. Notice, now we are evaluating the execution time by default. Evaluating the execution time should appear in all your Data-Structure codes. Write the code below and save it as “Hello.c”

**Code:**

**#include <iostream>**

**#include <string.h>**

**#include <bits/stdc++.h>**

**#include <sstream>**

**using namespace std;**

**int main(int argc, char \*argv [])**

**{**

**string A;**

**try**

**{**

**if (argc != 3)**

**{**

**throw(argc);**

**}**

**else**

**{**

**cout<<"Lets start the program"<<endl;**

**A = argv[0];**

**cout<<"Write Program Name: "<<A<<endl;**

**A = argv[1];**

**cout<<"Name of program to be repeated: "<<A<<endl;**

**A = argv[2];**

**cout<<"How many times: "<<A<<" x "<<endl;**

**// convertng string to number to int to repeatitions**

**int i, tau;**

**stringstream ss;**

**ss<<argv[2];**

**ss>>tau;**

**// printing how many times depends on user last input**

**for (int i = 0; i <= tau; i++)**

**{**

**cout<<"[ "<<i<<" ]"<<argv[1]<<endl;**

**}**

**}**

**}**

**catch(...)**

**{**

**cout<<"This program has 3 parameters: "<<endl;**

**cout<<"[1] Name of program like : [a b c].exe"<<endl;**

**cout<<"[2] Name of user you want to print like Ahmed: "<<endl;**

**cout<<"[3] How many times you want to print the name of user: "<<endl;**

**cout<<endl;**

**cout<<"for e.g: Typing Ahmed"<<endl;**

**cout<<"DSA Ahmed 3"<<endl;**

**cout<<"Will Ahmed print 3 times?"<<endl;**

**}**

**return 0;**

**}**

**Output:**



Fig: 02 CMD

**Q#03:** Input console application:

DST teaches you how to handle data using programming structures. The data encompassed within these structures is huge and goes way beyond asking the user to input details using the keyboard. Hence, reading input from an external file is fundamental. This lab teaches how to read an external file while storing the contents in another file. Focus on the comments, they are mentioned specifically for your understanding. As students are very keen in simply copy-pasting code, the following presents snap-shots (images of the running code) so that you write these codes yourself.

**Code:**

**#include <iostream>**

**#include <fstream>**

**#include <time.h>**

**#include <string.h>**

**#include <ctime>**

**#pragma warning(disable : 4996) // processor keyword**

**char \*input\_file\_name, \*output\_file\_name;**

**using namespace std;**

**int main(int argc, char \*argv[])**

**{**

**clock\_t start, middle, finish;**

**start = clock();**

**double execution\_time;**

**if (argc != 4)**

**{**

**cout<<"Author: [Your name here] \n"<<endl;**

**cout<<"Dear User! tell me about you how to use program. "<<endl;**

**cout<<"The program has 4 keywords: \n";**

**cout<<" [0] Code.exe "<<endl;**

**cout<<" [1] Name of file (input.text) "<<endl;**

**cout<<" [2] tau "<<endl;**

**cout<<" [3] Name of file (output.txt) "<<endl;**

**cout<<" [\*] Example: code.exe input.txt 20 output.txt "<<endl;**

**cout<<" [\*] The code will print the contents of input 20x in output.txt "<<endl;**

**}**

**else**

**{**

**cout<<"Author: [Your name here] \n"<<endl;**

**char date\_time[50];**

**time\_t t = time(NULL);**

**//ctime\_S(date\_time, 50, &t);**

**strftime(date\_time, sizeof(date\_time), "%c", localtime(&t));**

**cout<<" [\*] Today: "<<date\_time<<endl;**

**int len = strlen(argv[2]);**

**int tau = 0;**

**int p, o = 1;**

**for ( p = len - 1 ; p > -1; p= p-1)**

**{**

**tau = tau \* ((int)argv[2][p] - 48) \* o;**

**o = o \* 10;**

**}**

**input\_file\_name = argv[1];**

**output\_file\_name = argv[3];**

**ifstream myfile(input\_file\_name);**

**ofstream output(output\_file\_name);**

**if (myfile.is\_open())**

**{**

**string line;**

**while (getline(myfile, line))**

**{**

**for ( o = 1; o <= tau; o++)**

**{**

**output<<"["<<o<<"]: "<<line<<endl;**

**output<<endl;**

**}**

**}**

**myfile.close();**

**}**

**else**

**{**

**cout<<"[\*] Failed to open file: "<<input\_file\_name<<endl;**

**cout<<"[\*] Existing - check your files and folders and try again"<<endl;**

**return EXIT\_FAILURE;**

**}**

**finish = clock();**

**execution\_time = (double (finish - start)) /CLOCKS\_PER\_SEC;**

**cout<<"[\*] time taken: ["<<execution\_time<<"] seconds to complete task";**

**}**

**cin.get();**

**return 0;**

**}**

**Output:**

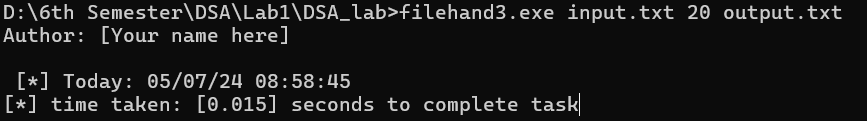


Fig: 03 CMD

A black screen with white text

Description automatically generated

Fig: 04 input.txt file

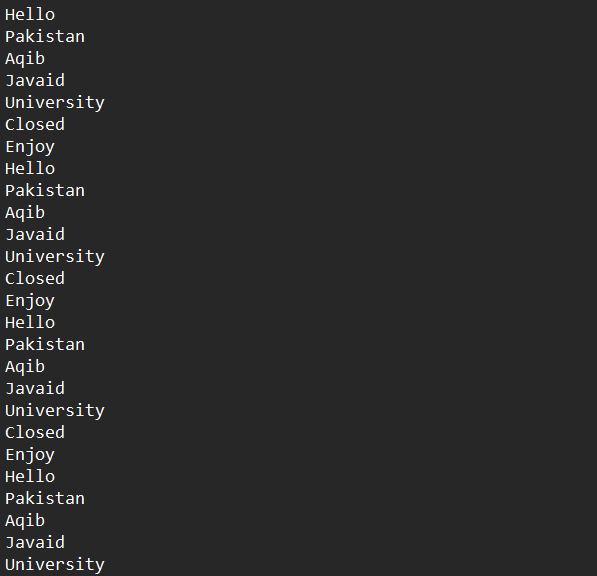


Fig: 05 output.txt file

**Methodology:**

Code 1: Defines a function fun() which waits for user input. It calculates the time taken for the function to execute and prints the result.

Code 2: Takes input parameters, including a program name, a username, and a repetition count. It then prints the user's name as many times as specified.

Code 3: Reads a file, duplicates its contents a certain number of times, and writes the output to another file. It calculates execution time and handles input errors.