UCF - CECS

Programming Assignment 1
8, 16, & 32 bit checksums

June 5, 2025

1 Checksum

In this assignment you'll write a program that calculates the checksum for the text in a file. Your program will take two command line parameters. The first parameter will be the name of the input file for calculating the checksum. The second parameter will be for the size of the checksum (8, 16, or 32 bits). The program must generate output to the console (terminal) screen as specified below.

1.1 Command line parameters

1. Your program must compile and run from the command line.
2. Input the required file name and the checksum size as command line parameters. Your program may NOT prompt the user to enter the file names. The first parameter must be the name of the file used for calculating the checksum, as described below. The second parameter must be the size, in bits, of the checksum. The sample run command near the end of this document contains an example of how the parameters will be entered.
3. Your program should open the input text files, echo the processed input to the screen, make the necessary calculations, and then output the *checksum* to the console (terminal) screen in the format described below.

Note
All of the test data files contain a termination character **LF** represented as a `hexadecimal '0A'`. This character is included in all the checksum calculations.

1.2 Checksum size

The checksum size is a single integer, passed as the *second* command line argument. The valid values are the size of the checksum, which can be either 8, 16, or 32 bits. Therefore, if the *second* parameter is not one of the valid values, the program should advise the user that the value is incorrect with a message formatted as shown below:

```
fprintf(stderr, "Valid checksum sizes are 8, 16, or 32\n");
```

The message should be sent to `STDERR`[1].

1.2.1 Format of the input file

The input file specified as the *first* command line argument, will consist of the valid 8 bit ASCII characters normally associated with the average text file. This includes punctuation, numbers, special characters, and whitespace.

1.2.2 Output Format

The program must output the following to the console (terminal) screen, also known as `STDOUT`:

1. Echo the text from the input file.
2. The echoed input text should be in rows of *exactly 80 characters per row*, except for the last row, which may possibly have fewer. These characters should correspond to the input text.
3. Print the checksum.
   - Remember to pad with **X** if the input data does not align with *checksum* size for the *checksum* calculation. For example, if calculating a 16 bit checksum, it could be necessary to add an additional **X** to arrive at an input file size of an even 16 bit size input. Likewise for 32 bits. *However, note that it may be necessary to pad with 1, 2, or 3 **X** characters for an even 32 bit size input.*
4. The checksum line should be formatted as follows[2] :

[1] Printing to `STDERR` can be accomplished using the followinge code:`fprintf(stderr,`*`normal printf format specifications`*`);` Java uses `System.err.println(...);`

[2] Where the variable checkSumSize is the checksum size of 8, 16, or 32, the variable checksum is the calculated checksum. Note that the checksums are masked to print the appropriate sizes such as two hex characters for 8 bits, 4 hex characters for the 16 bit checksum, and 8 hex characters for 32 bit checksum. The variable characterCnt is the character count of the input file and includes the terminating character LF or the hexadecimal value 0A.

```
        printf("%2d bit checksum is %8lx for all %4d chars\n",
                checkSumSize, checksum, characterCnt);
```

5. Remember to strip off the unnecessary bits. For example, if printing an 8 bit checksum, make sure to print out the 8 bits **only**. Likewise for 16 or 32 bit checksums.

1.3 Submission instructions

You must submit this assignment in **Webcourses** as a source file upload. Note that all submissions will be via Webcourses. The submitted programs will be **tested** and **graded** on **Eustis**.

1.3.1 Code Requirements

- Program name - the program name ***must be*** **pa01** in order to work with the shell scripts the graders use to grade your assignment. As mentioned before, the program **must be** named **pa01** for whichever language you choose. For example, **pa01**.c, **pa01**.cpp, **pa01**.java, **pa01**.py, or **pa01**.go.

- Header - the following *Header Usage instructions/comments* comment block should be at the beginning of the source file.

```
/*=============================================================================
 |   Assignment:  pa01 - Calculate the checksum of an input file given:
 |                 -> the name of the input file,
 |                 -> the checksum size of either 8, 16, or 32 bits
 |       Author:  Your name here
 |     Language:  c, c++, Java, go, python, rust
 |   To Compile:  javac pa01.java
 |                gcc -o pa01 pa01.c
 |                g++ -o pa01 pa01.cpp
 |                go build pa01.go
 |                rustc pa01.rs
 |   To Execute:  java   -> java pa01 inputFilename.txt checksumSize
 |          or    c++    -> ./pa01 inputFilename.txt checksumSize
 |          or    c      -> ./pa01 inputFilename.txt checksumSize
 |          or    go     -> ./pa01 inputFilename.txt checksumSize
 |          or    rust   -> ./pa01 inputFilename.txt checksumSize
 |          or    python -> python3 pa01.py inputFilename.txt checksumSize
 |                          where inputFilename.txt is the input file
 |                          and checksumSize is either 8, 16, or 32
 |         Note:
 |                All input files are simple 8 bit ASCII input
 |                All execute commands above have been tested on Eustis
 |        Class:  CIS3360 - Security in Computing - Spring 2025
 |   Instructor:  McAlpin
 |     Due Date:  per assignment
 +=============================================================================*/
```

- The following *Academic Integrity Statement* comment block should be at the end of the source file.

```
/*=============================================================================
 |     I [your name] ([your NID]) affirm that this program is
 | entirely my own work and that I have neither developed my code together with
 | any another person, nor copied any code from any other person, nor permitted
 | my code to be copied  or otherwise used by any other person, nor have I
 | copied, modified, or otherwise used programs created by others. I acknowledge
 | that any violation of the above terms will be treated as academic dishonesty.
 +=============================================================================*/
```

1.4 Program Notes and Hints

One possible breakdown to solve this problem is as follows:

1. Collect the command line input arguments and print them to the console. *Remember to remove or comment out this test code when running the testing scripts.*
2. Read the file and print it out to the console.
3. Adjust the output to print *80 characters per line*.
4. Calculate the 8 bit checksum. **Remember that the checksum is a running total with *no overflow*.**
5. Resolve the calculations and padding for both 16 and 32 bit checksums.
6. *Note this calculation is slightly different than the checksum calculation method discussed in lecture.*

1.5 Grading

Scoring will be based on the following rubric:

Table 1.1: Grading Rubric

Deduction	Description
-100	Cannot compile on *eustis*
-100	Your program does not successfully compile from the command line with one of these commands: C program: `prompt$gcc -o pa01 pa01.c` C++ program: `prompt$g++ -o pa01 pa01.cpp` Java program: `prompt$javac pa01.java` Python program: `prompt$python3 pa01.py` Go program: `prompt$golang pa01.go` Rust program: `prompt$rustc pa01.rs` Note: *If you are submitting a Java program, the class file must be named “pa01.java” and the class name must be “pa01”.*
-100	**Cannot read input parameters specified on command line**
-100	Cannot write output to `stdout`
- 90	The program does not run from the command line without error or produces no output.
- 70	The program compiles, runs, and outputs the input file, but crashes thereafter or produces no checksum output.
- 20	Fails to produce valid **8 bit** checksum
- 20	Fails to produce valid **16 bit** checksum
- 20	Fails to produce valid **32 bit** checksum
- 25	Does not have *Header Usage instructions/comments*
- 25	Does not have *Academic Integrity* statement
Start with 100 points and deduct per the schedule above	

1.6 Testing

1.6.1 Baseline

There are 6 baseline files included in the **ZIP** file. These files are `basetest.sh`, and the text input files, `i1.txt` thru `i5.txt`. (Additional test files are described in the next section.) The baseline filenames and their corresponding checksums are shown in the table below:

Table 1.2: Baseline Test Schema

Filename	8 checksum	16 checksum	Pad	32 checksum	Pad	Input (hex)	Input text
i1.txt	80	760a	-	760a5858	XX	760a	v\n
i2.txt	f7	80cf	X	76770a58	X	76770a	vw\n
i3.txt	6f	ee81	-	7677780a	-	7677780a	vwx\n
i4.txt	e8	f948	X	80cfd0d1	XXX	767778790a	vwxy\n
i5.txt	62	68fa	-	f081d0d1	XX	767778797a0a	vwxyz\n
Results were obtained using the command: *bash baseTest.sh pa01.|c|cpp|java|go|py|rs*							

1.6.2 Advanced Testing

There are eight input test files of significantly more varied content than used in the *Baseline Test Schema* shown above.

1. Every input file has a single line of text terminated by the hexadecimal character '0A' or the NEWLINE character.
2. Some input files are less than 80 characters long, others aren't.
3. More testing files are supplied than are used in the **pa01test.sh** script.
4. After uploading the testing shell script (and corresponding files) remember to execute the command **chmod +x *.sh** to grant execution privileges for the script.
5. The script is executed at the command line by the command **bash pa01test.sh pa01.c** or, alternatively **./baseTest.sh pa01.|c|cpp|java||py|rs** where the checksum program (*pa01*) filename has the correct extension for your submission. Valid language extensions are **.c** for C, **.cpp** for C++, **.java** for Java, **.go** for Go, **py** for Python, and **.rs** for Rust.

The expected output of the supplied test script is shown below.

```
 ./pa01test.sh pa01.java
Compile of pa01 succeded.
Case #1 - in10A.txt -  8 bit checksum
  -> program output matched expected
  -> Case #1 - complete
Case #1 - in10A.txt - 16 bit checksum
  -> program output matched expected
  -> Case #1 - complete
-> Case #1 - in10A.txt - 32 bit checksum
  -> program output matched expected
  -> Case #1 - complete
Case #2 - in17A.txt -  8 bit checksum
  -> program output matched expected
  -> Case #2 - complete
Case #2 - in17A.txt - 16 bit checksum
  -> program output matched expected
  -> Case #2 - complete
Case #2 - in17A.txt - 32 bit checksum
  -> program output matched expected
  -> Case #2 - complete
Case #3 - in18A.txt -  8 bit checksum
  -> program output matched expected
  -> Case #3 - complete
Case #3 - in18A.txt - 16 bit checksum
  -> program output matched expected
  -> Case #3 - complete
Case #3 - in18A.txt - 32 bit checksum
  -> program output matched expected
  -> Case #3 - complete
Case #4 - inRF2.txt -  8 bit checksum
  -> program output matched expected
  -> Case #4 - complete
Case #4 - inRF2.txt - 16 bit checksum
  -> program output matched expected
  -> Case #4 - complete
Case #4 - inRF2.txt - 32 bit checksum
  -> program output matched expected
  -> Case #4 - complete
Case #5 - inWC2.txt -  8 bit checksum
  -> program output matched expected
  -> Case #5 - complete
Case #5 - inWC2.txt - 16 bit checksum
  -> program output matched expected
  -> Case #5 - complete
Case #5 - inWC2.txt - 32 bit checksum
  -> program output matched expected
  -> Case #5 - complete
Checksum testing completed
```