|  |  |
| --- | --- |
| Project NumberCSC134 Project | Project 3 |
| Project Name | Morse Code Project |
| Project Filename | Firstname\_Lastname (example: *Jessica\_Smith\_Project3*) |
| Chapter Review | Chapter 5Use only the concepts covered in Chapters 1 - 5 to complete this project. Failure to do so may result in a 0 for the assignment.Students are expected to complete the practice programs before starting the project. |
| Points | 50 |
| Assistance | Instructors and teaching assistants have been available to assist with all practice work. Students are now expected to complete this project without assistance from others (this includes receiving assistance from individuals inside or outside of CPCC). Students should consider projects as non-proctored exams. Please review the academic integrity policy on your syllabus.Please note: students can continue to receive assistance with practice work up to 5:00 p.m. on the due date (review the late period on the syllabus). |
| Project Description | For this part of the project, you will add the functionality to translate a complete message into morse code and the functionality to process a data file containing multiple records. You will also modify your input validation code to give the user an unlimited number of chances to enter valid responses. Your program should continue to run until the user chooses a “quit” option from your menu.  **Instructions:**   1. Add an option to the menu to offer the user a choice of processing a data file. Add another option to quit the program. The menu should be the first thing displayed when your program begins executing. The menu format should match the one in the **Sample Output** on page 3. 2. Allow the user to continue making menu choices until they choose the option to quit. Always display the menu before prompting the user for a menu option. Properly handle an invalid menu choice. The user should have an unlimited number of chances to enter a valid menu choice. 3. Modify the code you wrote in Project 2 to validate the user’s input for payment amount to allow user an unlimited number of chances to enter a valid payment amount. 4. Modify the code you wrote in Project 2 to translate a single letter into a morse code. Your code should now be able to efficiently translate a complete message into morse code. (no code duplication). All white spaces in message should be translated into **3** corresponding white spaces in morse code.   **HINT:** Use the string class member functions at() and length(). See code example on Moodle.  If any part of the message cannot be translated into morse code, instead of displaying an error message like you did in Project 2, your code should display the untranslated character.   1. Add functionality to create and display telegram bills by reading input from the ***TelegramData.txt*** text file. When the user selects the option to process a data file, telegram bills should be created and displayed for every record in the file without the user needing to do anything else. The ***TelegramData .txt*** file contains 5 records, but your program must be able to process an identically formatted file with any number of records without counting the records. You do not need to process payment information if this option is chosen by the user. 2. Save your **.cpp** file using the Firstname\_Lastname\_Project3 naming format. Upload in Moodle. Verify that your file actually uploaded. 3. **Bonus Pts (5pts)**. Follow all the instructions above. Add functionality to verify that the message entered by the user only contains valid letters. If it doesn't, then a descriptive error message should display and the user should be given an unlimited number of chances to enter a message that only contains valid letters. You must follow all these instructions to receive full credit. |

# Sample Output

|  |  |
| --- | --- |
| Welcome to Western Union Telegraph Company  1 – Process Telegram Bill  2 – Translate to Morse Code  3 – Process a Data File  4 - Quit  Enter your choice: 2  Enter a message: That’s it!  Translation: **- . . . . . - - ‘ . . . . . - !**  1 – Process Telegram Bill  2 – Translate to Morse Code  3 – Process a Data File  4 - Quit  Enter your choice: 3  Leslie Knope  1456 Plymouth Street  Pawnee, IN 47408 Amount Owed: $35.50  Tom Haveford  689 Lil Sebastian Avenue  Pawnee, IN 47408 Amount Owed: $11.00  April Ludgate  1123 9th Avenue  Wamapoke, IN 48034 Amount Owed: $18.00 | Jery Gergich 3124 Woodbridge Road  Eagleton, IN 47322 Amount Owed: $114.00  Donna Meagle  1200 Elysian Fields Blvd  Eagleton, IN 47322 Amount Owed: $73.50  1 – Process Telegram Bill  2 – Translate to Morse Code  3 – Process a Data File  4 - Quit  Enter your choice: 44  44 is not a valid choice  1 – Process Telegram Bill  2 – Translate to Morse Code  3 – Process a Data File  4 - Quit  Enter your choice: 4  Thank you. Closing program. |

# Grading Score Card: 50

|  |  |  |
| --- | --- | --- |
| **OBJECTIVE** | **DESCRIPTION** | **POINTS** |
| Program Interface Well Designed | Menu is the first thing displayed when the program runs. Menu contains all the required options and the format matches the one in the **Sample Output**. Menu choices are properly processed and validated. | 5 |
| Program continues to run until the user selects the option to quit. The menu is always displayed before user is prompted for a choice. | 10 |
| User Input Validated | User input for payment amount is correctly validated. User is given an unlimited number of chances to enter a valid payment amount. | 5 |
| File is Read Correctly | File is correctly read and processed according to the instructions. Program can read similarly formatted files containing any number of records without counting the records. | 10 |
| Required Value(s) Correctly Calculated | Program correctly translates a complete message into morse code using an efficient algorithm. No code duplication. | 15 |
| Documentation and Programming Style | * Multi-line comment present at the beginning of the program that contains: your **Name**, **Date** and **Purpose** which fully describes what your program does. * Adequate descriptive comments used throughout program. There must be at least four comments to receive credit for this step. * Proper indentation and spacing used throughout program. * Proper naming conventions used throughout program. | 5 |
| **TOTAL** |  | **50** |
| **Bonus Points** | Functionality was added to verify that the message entered by the user only contains valid letters. If it doesn't, a descriptive error message displays and the user is given an unlimited number of chances to enter a message that only contains valid letters. | **+5pts** |
| **Point Deduction** | Program does not compile without errors. | **-20** |