Grant Combs

Dr. Jeff Horn

CS480

1 December 2021

Cipher Encryption and Decryption

Introduction

During the past few years, encrypting and decrypting ciphered messages has become a shared hobby between me and my friends. Writing a message, taking the time to encrypt it by hand, and then sending it in the mail provides an extra layer of connection that digital means of communication cannot always provide. This process has become a creative way for us to stay in touch across long distances, and presents a challenging puzzle when attempting to decrypt a ciphered message from another person. When presented with the opportunity to write a program of my choice, it seemed appropriate to write one that had to do with one of my hobbies; therefore, I decided to write a program that would automate the process of encryption and decryption.

The premise of the program is simple: the user can either choose to encrypt a message or decrypt one. If they choose to encrypt a message, the user enters the plaintext, then fills out the parameters for the cipher that they want to use, and clicks “Encrypt”. To decrypt a message, the user enters the encoded message, then selects what cipher they think was used, then clicks “Decrypt”. The user can select from a variety of different ciphers, both monoalphabetic and polyalphabetic.

The purpose of the program is to speed up the process of amateur codebreaking. While it does not include every cipher in existence, it does contain several of the most commonly used ones. In addition to aiding amateur codebreaking, the program also serves to further my understanding of how each individual cipher functions, and how to best go about deciphering messages.

About the Project & Challenges

For this project, I wanted to create a program that could run on both my university-provided laptop and my personal Windows machine. I also wanted to give the program an actual user interface, something I had never done before. To accomplish both of these, I decided to create a Windows Power Form Application in Microsoft Visual Studio 2019. Microsoft gives programmers the option to write Windows Power Form Applications in either C# or Visual Basic; I picked C# as it is more closely related to programming languages I have already learned.

Coming primarily from a university background of C++ and Java, I faced a few challenges learning the C# language. For example, prior to this project, I had never learned a programming language purely from self-teaching. Given that I had already had three and a half years of programming experience, this was not quite as difficult as I envisioned it, but was still a challenge. I started by giving myself a quick overview of the language by watching a few introductory videos on YouTube. Then, I programmed a basic calculator with the help of some tutorials online, which helped familiarize me with the syntax, basic feature-set and quirks of C# as well as the process of building a Windows Power Form Application in Visual Studio, and how

the program connected the user interface of the program to actual code that could be executed when certain actions were carried out by the user.

The program itself is designed to be as simple and straightforward as possible. When a user is ready to encrypt or decrypt a message, they can select a cipher from a dropdown menu. When the user selects a cipher, a method is called that changes the parameters visible to the user. The user is then able to modify those parameters and enter their message, whether plaintext or the encrypted message. When the user clicks “Encrypt” or “Decrypt”, there is a switch that gets the name of the cipher that is currently selected in the dropdown menu, which then calls the appropriate encryption/decryption method for that cipher. The simpler ciphers, such as the Caesar and multiplicative ciphers, have relatively simple methods, whereas more complicated ciphers, such as the Hill Digraph and Playfair ciphers, have more complicated methods. The latter also have helper methods that break up the code into modular parts: one that creates digraphs based on the user message, and one that multiplies matrices. This allows these portions of code to be used by multiple other methods.

The program also contains a great deal of error checking, something that I have not always included in my university programs. However, when building a program that is made to be used by an end user, I needed to make sure that there was no way to make the program crash, which required a lot of attention to what mistakes a user could potentially make to cause problems. When an encryption or decryption method is called, the OptionRequirementsMet method is called, and the necessary parameters for that cipher are included in the arguments passed to the function. The function then returns a list of objects, which are the user interface elements that have not been appropriately filled out by the user. The method

DisplayRequirementError then displays an error message containing all of the errors detected in the OptionRequirementsMet method.

When I began my project, it was my impression that C++ and C# were similar languages. However, I soon realized that C# worked quite differently, and I discovered that there were many parts of the language that were much more intuitive to me- for example, I found it very useful that you can pass a variable "reference" to a method as opposed to having to pass a pointer to that variable. Essentially, C# references demonstrate a slightly higher level of abstraction that pointers, as directly working with pointers/memory addresses is referred to as "unsafe" and is generally discouraged. I learned that C# also contains several other useful features such as automatic garbage collection, and the ability to use a string as the decision variable in a switch, which I used in my program. Thus, learning C# during the project offered me a unique learning experience that gave me exposure to a new set of programming features and gave me the opportunity to learn a programming language on my own terms.

What went well, and what didn't

Throughout the duration of my project, there were aspects that went really well, and areas in which I struggled. Starting with the positive aspects, one of the easier parts of the project was writing the code that interacted with the user interface. Writing methods that ran when a button was pushed, or a drop-down menu item was clicked, was relatively simple compared to the rest of the program. Another aspect of the project that went fairly well was the methods that handled encoding messages. The monoalphabetic cipher methods were the easiest to write, given that each encrypted letter is essentially the result of a function of the input letter. Some of the

polyalphabetic ciphers, such as the Hill Digraph and Playfair ciphers, were more difficult to implement given their relative complexity. However, after many hours spent visualizing (and later debugging) how to convert the encryption process to computer code, I managed to successfully implement these ciphers.

Programming the methods that handled decryption with known parameters was also relatively simple to implement, as these were merely a reverse of the encryption methods that I had already written. What challenged me the most was writing a method that would be able to decrypt messages when the user did not know the parameters with which a ciphered message was encoded. When you do not know exactly what words you are searching for, the task of automated decryption becomes difficult. I asked myself questions like, "How does the algorithm decide when it thinks it found a match to the cipher?" and "What is an appropriate way to rank potential cipher solutions?" The solution I devised runs through all possible combinations of a cipher's parameters and assigns a score representing how many matches to real words it found. The program does not have an unlimited vocabulary—instead, it draws from a list of the most common words in the English language. The program will then display the solutions with the highest score. This approach, while functional, does not provide great runtime. In order to optimize runtime, I was able to tweak the number of words that the algorithm searched for—the minimum number of words that I tried searching for was 100, and the maximum was 10,000 words, which was the size of the full list. This led to an interesting series of tests to determine the optimal dictionary size.

Not all of the project went as planned, though. I originally intended for my program to only be for decrypting ciphered messages; however, as development began, I decided that I wanted the program to be a more well-rounded cryptography program that would be useful for

both encryption and decryption. After all, while there are many already existing programs that attempt to decrypt ciphered messages, there are not many that encrypt messages. If I had the opportunity to do the project over, I think that I would investigate writing the program in another language, particularly one that would compile and run efficiently on Linux. However, since I had no experience in designing user interfaces, Visual Studio's Windows Power Form Application design window was very helpful, and was a solid first step in designing real user interfaces. Another aspect I would perhaps do differently is my approach to decrypting messages when the user does not know the parameters with which a message was encoded. While my implementation works, there is much to be desired in terms of performance. In order to have a reasonable runtime, I had to reduce the number of words that the program searched for. Lastly, my project could always be expanded upon in regards to the number of ciphers included. Given that there are virtually an infinite number of ways to encrypt a message, it makes the number that I implemented seem relatively insignificant.

Conclusion

Overall, writing my program gave me a multitude of learning opportunities ranging from learning a new language to building a program with a user interface to learning how to approach decryption algorithms. The program provides users a basic tool to automate amateur codebreaking in a straightforward user interface and overall gave me a better understanding of ciphers, encryption, and decryption techniques.

Works Referenced

Novig, Peter. *The 10,000 most common English words in order of frequency, as determined by n-gram frequency analysis of the Google's Trillion Word Corpus. Natural Language Corpus Data: Beautiful Data.* Linguistic Data Consortium and Google, 2009. Accessed 5 November 2021. Dataset.