

Reverse Engineering and Analysis of the Mystery Code

Analysis Procedure

Figuring out how to read the code was a multi-step process that required the use of various tools and techniques. Each step helped in the process to more clearly get a visual of what was going on in the code.

Steps:

1. **Copy strings from function calls into their own variables and replace the strings in the function calls with their respective variable names.** This helps to shorten the code by separating the functional parts of the code from the static values.
2. **Remove all whitespace from the functional code and turn the return statement in to one line.** Doing this was so that there wouldn't be any confusing tabs or spaces to throw me off for following steps.
3. **Use parenthesis to clearly and visually encapsulate the parts of the code and make it easier to see where parts of the code lie with respect to other parts.** This step requires compiling multiple times to make sure each step retains the same logic and doesn't generate any errors. This step adds many redundant parenthesis around various parts of the code, but makes it clear where each part begins and ends.
4. **Adding the whitespace to the code.** This is to make the code look more readable and like traditional code with proper indenting. Whenever I would run in to the beginning of a set of parenthesis, I would make a new line and add an indent. Whenever I came across the end of a set of parenthesis, I would reset the indent. After this I could remove the unneeded parenthesis.
5. **Converting the ternary operators in to their keyword if-else counterparts.** This step helped with the debugger recognizing each step in the code's algorithm. I would add a return to each expression I saw as necessary. The one thing that was slowing down my progress was the commas near the first part of the code:

```
t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?main(2,_+1,"%s %d %d\n"):9:16
```


After a little research, though I figured out that the return statement only returns the value of the expression after the last comma. Anything before that is just to create side effects. Similar to having multiple lines in code where only the last line has a return statement. So I then structured the code based on that to just be an if statement with three independent if statements executing in order.
6. **Read through the code and rename the variables according to what I thought their purpose was.** This helped to make the code more readable so I could easily recognize what the code was doing and what was being changed or used based on the names. I also changed the way some of the logic was written so that it would be the same logic, but more straightforward about what it was doing. Changing things like `if(!0<t)` to `if(1<t)`, since `!0==1` in c, or even `if(t>1)` made it much easier to read for me.
7. **Adding comments for what the logic was doing.** Many of the if statements were executing only when t was within an expected range. I used comments to spell out what parts of the code to look at when t was being passed. This made the process of tracing the code much easier. I

also left comments about what behaviors certain parts of the code would produce. Making it much easier to resolve what would happen under certain conditions in the code.

8. **Run the code through a debugger.** This was the final step. I used this after I had figured out most of the code and there were just a few things I was missing in my understanding of the code as well as to verify that I understood the code. I took the code step by step to see what would happen to the variables placing breakpoints at certain locations and repeatedly continuing to see what had updated. This helped me confirm how exactly the code was working and make sure my trace was correct.

The Program/Recursion Trace

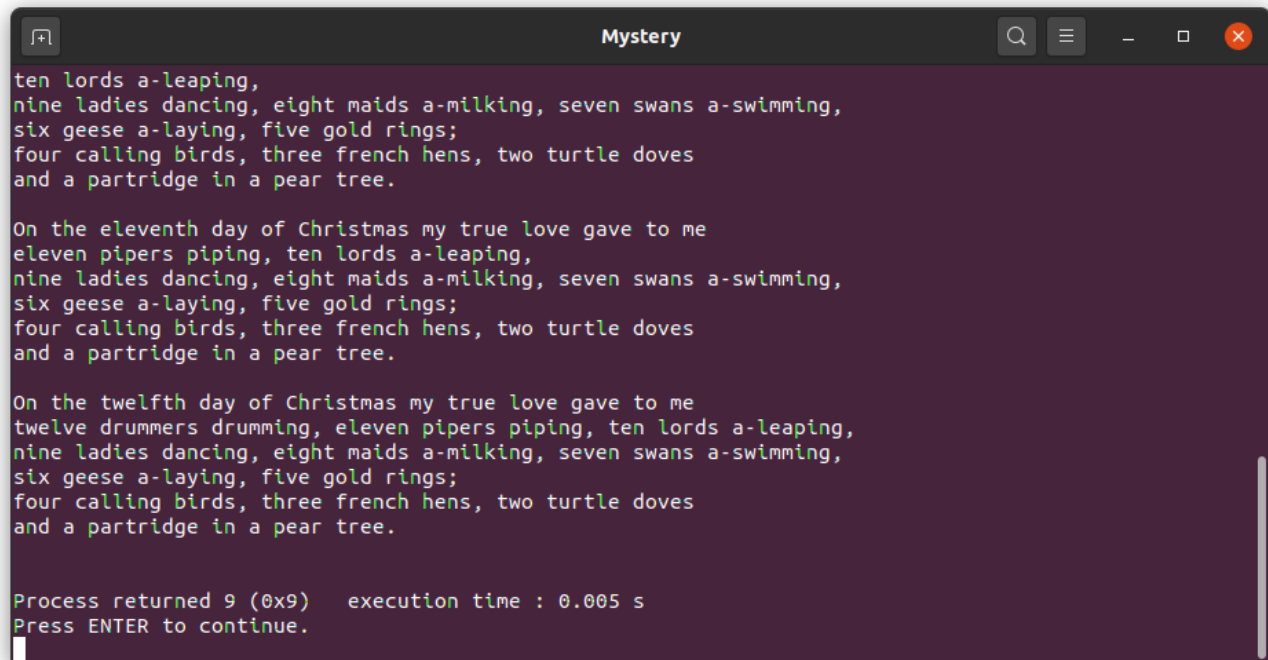
The program starts off with `t` initialized to 1. The other values are random, but they don't matter. With `t` as 1, the first recursive call is made as `main(2,2,str3)` setting up the `t` and the `_` as 2 and using the current string value. With `t` now as 2, the next functions that are called are `main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a))` from the innermost `main` to the outermost. Since all `t` values for these functions are less than -72, the next function each of them will be calling will be `main(param2,param1,cypherString)`, which will swap their `t` and `_` values. The first swapped call will be `main(0,-86,strPtr+1)`. Since `t` is now 0, this will invoke the innermost call of the next set of functions `main(0,main(-61,*a,referenceString),a+1)`. This will set `t` as -61, and set it up with the character in its third parameter as a `_` value and the reference string to compare and convert the character in to the correct format. It will then continuously call `main(-65,_,a+1)` until `a` matches `_`. Once a match is found it calls `putchar(a[31])` which just outputs the character 31 units to the right to the console. It will continue to do this until it has converted all characters up to the next `'` in the string. It then works its way back to the outer functions until it reaches `main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a))` again. It adds each string to the last using the `_` value as an index for the block of characters indicated by the `'` until it ends up with "On the nth day of Christmas my true love gave to me". If the `t` value is less than the `_` value, it then works on the `main(t+1,_,a)` function. This function will work from the top continuously calling itself after the previous function calls have been made, if applicable and work backwards once `t` and `_` are equal. The next function call is made in an if statement. `main(-94,-27+_,a)!=0` is called which then invokes `main(_,t,a)`, flipping the `t` and `_` values. With `t` now being -27, this causes the program to invoke `main((*a=='')+t,_,a+1)`, which repeatedly calls itself, incrementing the `a` value pointing at the string and incrementing the `t` value whenever it comes across a `'` separating character until `t==0`. Once `t==0`, it will then call the nested function `main(0,main(-61,*a,referenceString),a+1)`. This will have the same behavior as previously mentioned, returning either a 0 or non-0 value which the computer then checks. If it is not 0, the computer short-circuits and the program returns an error. Otherwise it also checks if the `t` value is 2. If so, the program checks if the 12 iteration limit has been reached. If so, the program returns with a success, otherwise the program invokes `main(2,_,a)`. This will start the program over again with an increased `_` value to increase the number until the value returns a success.

Recursive trace (visualized):

```
main(t=1,_,a)
main(2,2,"%s")
main(-86,0,"%s")
main(0,-86,cypherString)
```

```
main(-61,cypherString[0],referenceString)
main(-65,cypherString[0],referenceString[i++])
// The above recursion function call is repeated until cypherString[0] == referenceString[i]
putchar(referenceString[i+31])
main(0,referenceString[i++],a+1)
// The above recursion function call is repeated until the next '/' character is encountered
main(-87,-1,"On the "+"") // This call is the same as the former but it is used to find the day string
main(-79,-13,""+"On the nth") // This calls out the last part of the first line
main(3,2,"")
// This call is repeated until t == _
main(-94,-25,"")
main(-25,-94,cypherString)
main(-65,-94,cypherString[i++])
// The above recursion function call is repeated until cypherString[0] == referenceString[i]
putchar(referenceString[i+31])
main(2,3,"%s %d %d\n")
// Repeats the above recursive call until _ is 13
// If _ reaches 13, it returns a success, otherwise it fails
```

What is This Code?



```
ten lords a-leaping,  
nine ladies dancing, eight maids a-milking, seven swans a-swimming,  
six geese a-laying, five gold rings;  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.  
  
On the eleventh day of Christmas my true love gave to me  
eleven pipers piping, ten lords a-leaping,  
nine ladies dancing, eight maids a-milking, seven swans a-swimming,  
six geese a-laying, five gold rings;  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.  
  
On the twelfth day of Christmas my true love gave to me  
twelve drummers drumming, eleven pipers piping, ten lords a-leaping,  
nine ladies dancing, eight maids a-milking, seven swans a-swimming,  
six geese a-laying, five gold rings;  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.  
  
Process returned 9 (0x9)   execution time : 0.005 s  
Press ENTER to continue.  
█
```

When compiled and run, this code ends up printing out the lyrics for the song 12 Days of Christmas. The contents of which can be found in the file named 'Output'. In the form that it was given, this code is about 1/3 the size of its output. It could even be further shortened by taking out the decyphering portion of the code and storing the text in plain text. This program also in its original form has no storage apart from the stack calls and the parameters stored in it. There are no variables defined within the function definition, only in the function signature. What this program essentially does is take the pre-programmed strings, passes the string with the data through the cypher string and outputs that to the terminal. The stack-loading caused by the recursion is used to keep track of and build up the separate strings meant for the output, giving the effect of printing out a new part of the string on top of what was previously printed. It is also used as a replacement for iteration in cases where a search needs to occur.

Explanation of Assignment Contents

Aside from this essay I have included four files in the zip file to help explain my process. The files are as follows:

Formatted.c – Mostly the same file as the original, just with parenthesis encapsulating all related expressions and whitespace making hierarchies clearer.

Updated.c – The fully updated source, with added comments and updated variable names to make the file more easily readable. This was a big step for me in understanding the code.

Output_No-Shift – The output, but with the 31 unit shift removed. So the program just prints out the string as if it's plain text. This helped me verify what the program was doing by matching the characters being output by the program to the characters on the original string.