Results

Survey 855573

Number of records in this query:	54
Total records in survey:	54
Percentage of total:	100.00%

Field summary for lastsurvey(SQ001)

Did you take part in one of the previous decompiler surveys? [Decompiler Survey 1 - October 2020]

Answer	Count	Percentage
Yes (A1)	18	33.33%
No (A2)	31	57.41%
No answer	5	9.26%

Field summary for lastsurvey(SQ002)

Did you take part in one of the previous decompiler surveys? [Decompiler Survey 2 - April 2021]

Answer	Count	Percentage
Yes (A1)	24	44.44%
No (A2)	26	48.15%
No answer	4	7.41%

Field summary for c

How much time (approx.) did you spend working with C code? (Development, Reviews, Learning, ...)

Answer	Count	Percentage
None (C0)	0	0.00%
A few hours (C1)	2	3.70%
Several days (C2)	14	25.93%
More than a year (C3)	25	46.30%
On a regular basis (C4)	13	24.07%
No answer	0	0.00%

Field summary for reversing

How much time did you spend reversing executables before?

Answer	Count	Percentage
None (R0)	2	3.70%
A few hours (R1)	11	20.37%
Several days (R2)	18	33.33%
More than a year (R3)	12	22.22%
On a regular basis (R4)	11	20.37%
No answer	0	0.00%

Field summary for ranking [1]

Please rank the decompiled code presented.[Ranking 1]

Answer	Count	Percentage
Clove (ida)	7	12.96%
Cumin (ghidr)	3	5.56%
Chilli (dewol)	44	81.48%

Field summary for ranking [2]

Please rank the decompiled code presented.[Ranking 2]

Answer	Count	Percentage
Clove (ida)	37	68.52%
Cumin (ghidr)	8	14.81%
Chilli (dewol)	9	16.67%

Field summary for ranking [3]

Please rank the decompiled code presented.[Ranking 3]

Answer	Count	Percentage
Clove (ida)	10	18.52%
Cumin (ghidr)	43	79.63%
Chilli (dewol)	1	1.85%

Field summary for positive

Which aspects in the decompiled codes above is especially favorable to you?

Answer	Count	Percentage
Answer	52	96.30%
No answer	2	3.70%

ID	Response
1	switch case helps understanding conditions more easily
3	The switch-statement of Chilli is nice especially vs the " " indentation of Cumin. Clove is the only one that recoginizes that 's' is used as an array and not a single value.
6	Assuming they are logically equivalent (not heavily scrutinized), elimination of deep nesting of if/else is most favorable. Use of X[i] notation as opposed to *(X + i) might be preferable too. Identification of a for loop to replace a while loop is useful too.
7	1. Chilli: I prefer the constructed switch-case, as it reduces code complexity and improves readability 2. Clove: The RSP/RBP annotations for variable declarations are useful
8	overview: the less indentation the clearer it gets + cleverly reconstructing common code
10	Switch statements in chilli.
12	Note: Chilli/Clove are pretty close. Clove - It's quite immediately obvious what each branch is doing, because of a) identifying s as being accessed as an array and b) directly indexing with strlen(s) makes it clear what is going on. That made it quite easy to make a good guess at what the function is doing quickly. Chilli - The switch statement makes the control flow much more obvious. Also, the lifting of the loop conditions into the head of the for loop also helps make the purpose of the loop more immediately apparent. Cumin - The lift of the loop condition into the loop head is useful.
13	The close relation to the (probably) original C Code, with a switch statement instead of nested if-else blocks in "Chilli".
14	Switch cases in Chilli makes code much more readable.
16	readability: resolved types, shortness, levels of nesting
17	* Low if-nesting * Proper datatypes
18	variable name numbering
19	Chilli had cleaner and more human readable codes. This makes analysis much easier as compared the one seen in Cumin.
20	Propper switch-case extraction
22	The use of switch case to simplify the code.
23	Easier control-flow analysis (e.g. prefer switch statements rather than convoluted if-else statements)
26	Chilli: Use of switch statement. Use of for loop instead of while loop makes the condition more clear as well. The lack of type casting also makes it looks cleaner (although it would be useful for user to enable typecasting to be shown). Clove: Deduction of the "s" as char array and then indexing it automatically is helpful.
29	Chilli is able to identify and decompile as a case-switch. Clove however provides good detail on declaration of variables which may be useful It would be good to have the option to display constants either as hex or decimal depending on preference.
30	Switch makes things easier to read for me, compared to repeatedly nested if-else.
31	- Appropriate use of switch-case - Omission of redundant checks - Omission of implied typecasts - Use of /=, which is quite succinct
32	readability of chilli, using switch case instead of if else resulting in less indentations and also easy to read which conditions will trigger which code path.

34	type information and avoiding deeply nested ifs
37	Data types, sufficient verbosity e.g. switch case statement not expanded into if-else
38	- using a switch statement - using decimal values - using a while loop - indexing arrays instead of using offsets - using less type casts
39	Use of switch-case instead of endless if/else if
40	The decompiled switch-case, and i find "var_2" easier to read than v2. Also the local vars in chilli help the understanding I like that clove displays the stack offsets of the variables
41	the switch statement ist easier to understan
42	Chilli has the least amount of nesting. For this example less nesting means much better readability. Clove contains less typecasting than Cumin.
43	switch-case instead of nested if-else for loop instead of while loop
46	switch case instead of nested if else statements
47	Modulo detection in Clove/Chilli Concise loop expression in Clove/Cumin Inlined writes to s in Chilli
48	Easy to understand at a look
50	Using switch-case instead of nested if statements, as it can get very messy.
51	Readability, switch case statement instead of large nested conditional statements
52	Switch statement Array recognition Comments mapping variables back to memory locations
53	I like how chilli looks like a proper source code. However, i do trust a decompiler that does less analysis more.
54	I liked how Chilli was able to recover the switch statement. Clove, by not assigning the result of strlen(s) to an intermediate variable, was able to produce a decompilation with less visual clutter.
56	Neatness; able to quickly deduce there are switch statements and the values for each case.
57	Shorter expressions tend to be more easy to read. The switch-case statement appears to be more a lot clearer than the nested branches.
58	switches in place of a huge and nested if-else block makes the decompiled code more readable.
59	Chilli
60	switch case in chilli
62	Especially the 'Chilli' decompiler, produced a clear and easy readable code, which wasn't too nested due to the use of switch-case statements.
63	Chilli: The switch case is much easier to read than the nested if/ else/ else if structures. Chilli and Clove: The instructions before entering the loop where optically clearly distinguishable from the loop.
64	Clear structure based on the switch allows better comprehension than nested if statements.
78	easy to follow along structure
79	The use of switch case from Chili makes the code cleaner and readable.
80	deeply nested code is hard to read, hence I enjoyed the one with switch. Apart from that arithmetic was displayed a litter bit better in Chilli than Clove.
82	presenting as case, instead of massive loops. Showing the logic of the loop condition.
84	more human readable, closer to high-level programming code
85	Switch statements are easier to comprehend compared to if-else
86	switch case

Field summary for negative

Which aspects of the presented decompiled functions do you deem unhelpful?

Answer	Count	Percentage
Answer	50	92.59%
No answer	4	7.41%

ID	Response
1	hard to follow the heavily nested if/else, cumin with many unnecessary variables
3	Chilli could have realized 0x2710 is a power of 10. Which the other decompiler did, *wink wink*. Chilli is the only one that does not produce valid c. Cumin for some reason does not display the string in scanf and has redundant else branches increasing indentation.
6	THe use of undefined8 is not as helpful.
7	1. /* format */ annotations 2. Cumin did not resolve the constant format string used in the __isoc99_scanf call 3. Cumin introduces WAY too much cyclomatic complexity. I think it's hard to follow the control flow there
8	n/a
10	insanely deep nesting. weird for loop (cumin)

12	Clove - if/else branching is messy to read, it is not immediately obvious how or when the loop terminates until reading the end of the loop
	Chilli - The branch of each switch statement is a bit harder to understand, and might be considered somewhat misleading - consider the following:
	var_3 = &var_2;
	var_5 = strlen(var_3);
	*(&var_2 + var_5) = 0x30;
	break;
	The var_2 is typed as long, so it looks like &var_2 + var_5 is indexing &var_2 as a long * using the strlen character count, and then dereferencing it as a long, which doesn't seem to make sense in this context.
	Cumin: I am not sure what 'undefinedn' types are supposed to be (I assume it is a n-byte type), not printing in the short constant string in scanf is a little annoying, the modulus operation on local_18 was not simplified, the control flow within the loop is harder to understand without the switch statement, branches of the switch statement are slightly less concise.

13	Many variables and nested if-else blocks.
14	Nested if-else conditions Some parts of the code would be easier to understand when the integers were in decimal instead of being converting to hex
16	levels of nesting, unresolved types, pointer arithmetic
17	* Chilli: Pointer arithmetic and dereference instead of indexing operator * Cumin: Deep nesting and unhelpful datatypes * Clove: Nesting
18	nil
19	While loops as seen in Clove were unhelpful and impedes analysis when determining the flow of the program.
20	to many local variables, deeply nested control structures
22	The use of &local_68 + sVar1 in Cumin seems largely unhelpful.
23	Referencing static data through global variable (Cumin: __isoc99_scanf(&DAT_00403259,&local_18);). Would rather not click on the variable to find out what it actually is.
26	Cumin: Heavily nested if statement makes it hard to read. Showing DAT_00403259 instead of the string literal "%lld" means user have to waste time and check the variable and fix it (presumably to make it read-only).
29	Large nested if-else
31	- Omission of redundant checks must be done carefully, in case there may be some vulnerabilities in the checks, e.g. overflows
32	if else instead of switch case

34	deeply nested ifs
37	Data types as WORD/QWORD, switch case expanded into if-else
38	- using long nested else-if constructs - using hex values (when not optimal for readability like e.g. for addresses) - using many type casts - array offset arithmetics - no or undefined variable types
39	Refer to this: https://i.imgur.com/BtjZedW.jpg
40	in clove the statements in the form of "*(_WORD *)&s[strlen(s)] = 56;", so as a one-liner, are very hard to read. The long else-if chain is inferior to the switch case.
41	the exessive type casts in Cumin are confusing, especially with casts to "undefined"
42	Chilli mixes base-10 and hex-representations of integers for their cases. I prefer a consistent usage of only one of the two. Chillis representation does not contain information about stack positions of variables, which can be quite useful in cases where the decompiler output is not completely correct.
43	clove uses instructions like *(_WORD *)&s[strlen(s)] = 70; which are not readable at all big amount of undefined local variables (in contrast to chilli)
46	nested if else statements
47	The /* format */ annotation in Clove's output Messy chain of if/else in Cumin/Chilli due to lack of switch detection
48	Too many nested else and if
50	Naming of variables like "local_10".
51	Unncessary nested "" conditions just to check for another if condition for "==" , such as the one below else if (v11
52	undefined8 Having to add offset to pointer and dereference (instead of array access)
53	Nil. I do understand why all the aspects are needed and can be helpful even though they are unintuitive.
54	Cumin's decompilation is probably closest to what it looks like in disassembly - a lot of branches. Not very useful though. Likewise with Clove, but Clove does it much better with a cleaner by using lesser intermediate variables.
56	Too many unnecessary casting information
57	Long expressions (containing cast operations) are more difficult to read. The nested branching is somewhat cluttered and more difficult to follow.
58	the decompiled functions here are generally okay, except for the nested if-else blocks.
59	Cumin
60	too many indentations
62	Declaration of numerous local variables, which unclear usage. E.g., 'Cumin' defined the following variables: undefined8 local_68 undefined8 local_60, undefined8 local_58, undefined8 local_50, undefined8 local_48, undefined8 local_40, undefined8 local_38, undefined8 local_30, undefined , cal_28, long local_18, int local_10 uint local_c. But from these variables, only a few (= local_68, local_18, local_10, local_c) were actually used in the code above. This results in cluttered code, with more hard to understand.
63	Clove: The nested else if statements are hard to read. Cumin: The nested if statements were not as easy to read as the switch case structure in Chilli. The many variables were a bit confusing.
64	given nested if statements beyond degree 2-3, any alternative cases (especially on the same level but lines further down in the code) become hard to follow quickly.
78	too many if else conditions
79	Too many if else statements from Cumin which makes the whole code messy and harder to read & understand.
80	lots of redundant local variables
82	Cumin view, logic flow difficult to understand at first glance.
84	unnecessary casting and convoluted computation
85	Too many layers of nested blocks of code.

Field summary for CopySimple

Please consider the following two code snippets, which both represent the same semantics with a different structure. In Snippet 1, /* Block #10 */ is copied to achieve a simplified structure, while in Snippet 2 it occurs only once. Snippet 1 Snippet 2 Which structure do you prefer in general?

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	14	25.93%
Snippet 2 (Not copying /* Block #10 */) (A2)	31	57.41%
No preference (A3)	8	14.81%
No answer	1	1.85%

Field summary for ConditionIfElseInstr(SQ001)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 1 (1 Instruction)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	42	77.78%
Snippet 2 (Not copying /* Block #10 */) (A2)	4	7.41%
No preference (A3)	8	14.81%
No answer	0	0.00%

Field summary for ConditionIfElseInstr(SQ002)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 2 (5 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	10	18.52%
Snippet 2 (Not copying /* Block #10 */) (A2)	36	66.67%
No preference (A3)	8	14.81%
No answer	0	0.00%

Field summary for ConditionIfElseInstr(SQ003)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 3 (15 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	3	5.56%
Snippet 2 (Not copying /* Block #10 */) (A2)	49	90.74%
No preference (A3)	2	3.70%
No answer	0	0.00%

Field summary for ConditionIfElseFree

Please explain your choices. Which criteria guide your decision?

Answer	Count	Percentage
Answer	49	90.74%
No answer	5	9.26%

ID	Response
1	the length of the structure matters, but the main reason to include it not twice is the extra step it adds to understanding which configurations use the specified paths
3	At some scoping levels duplicating a ret, or a jump to the epilog seems worthit, otherwise it depends on how much code is around it. A return statement inside of a while-statement is often confusing and in the given example the scope-depth is manageable, so i'd vote for no duplication.
6	Am assuming code duplication helps in writing a "cleaner-looking" while loop. The overall guiding principle is to prevent analysis of additional code that appears elsewhere too.
7	I'd prefer snippet 1 for relatively small code blocks. . However, the larger the block, the harder it would be to visually distinguish between identical code and code with slight differences.
8	overview: the less code one sees, the easier it gets to reason about the code
10	for short snippets context is clear even if they are copied, leading to cleaner code. copying longer sequences might lead to confusion
12	Primary factor is how close it it to code that I think a human might write. I think except for very short blocks like Option 1 it feels more natural to not duplicate the code, even if it makes the control flow a little bit more complex. Having less code to read is generally good as well, as now I do not have to figure out whether two blocks of code separated by large amounts of other code actually do the same thing or not. At the cost of slightly more complex flow I think this makes reversing easier.
13	To copy one return statement is just fine, but if it's more than one or two lines, I think it is confusing, and it bloats the decompiled code.
14	With only 1 return instruction, it's more readable to understand that the function terminates given certain conditions. However, when there's additional logic, probably a shorter code (with no copying) is more readable.
16	If the code gets too long, it's more difficult to see, that it's the same block of code, if it is copied. It's shorter and better to overview, if it's not copied as well. But it depends on the code itself. If the code itself gets more complicated and nested, if the block is not copied, the copied option may be better.
17	Preventing duplicated code, but a single return statement is not worth the more complex control flow.
18	Even if Snippet 1 has a simplified structure, a lengthy and repeated "Block #10" would not make it easier to read as compared to Snippet 2.
19	Single line codes will take less effort in terms of analysis, in terms of the example, a return shows the end of a pathway quickly, allowing for (targeted) quick dives of code snippets as compared to full code analysis. Longer codes (Option 2 and 15) will require more tactful approach as this functions occurring twice in a single function may throw off the analysis and waste valuable time re-analyzing a function that has already been dived through.
20	single statements are tolerable
22	Is there is too many instructions then it will over complicate things if we repeat it.
23	A simple return statement will be good in guiding control-flow analysis. However when there are additional n operations performed before the return, it may complicate/make analysis less intuitive. For example if there are huge chunks of operations done before the program return, I may forget what happens during a return and analyse the n operations again. In this case, having the /* Block #10 */ in a single place helps simplify my thought process.
26	Copying is fine as long as it is not too long. Forcing it to be a single block can make other code structure become harder to read. E.g. snippet 2's while loop contains several break statement which makes it harder to follow.

29	Once we've identified a unique block of code, repetition is fine since we can easily mark and understand the function. It's more important to have a simplified structure to better understand the overall flow.
30	For shorter snippets I prefer it to be copied since it helps me parse the flow better; forcing a break out of a while(true) just to handle it without copying seems a bit strange to me. I put no preference for longer snippet because it really depends on how long the other code blocks are as well; i think there's merit to keeping code short of the rest of the code is already super long.
31	- I prefer while-condition loops to while-true loops with many inner checks, as they are easier to interpret. - It is hard to compare longer snippets and notice that they are identical. Possible solutions: - Inline function - Goto label
32	If the same block of code can be presented without copying (so there is only one block to process) it is much simpler to understand the code flow and logic. Having two (or more) exact copies of a block of code in a function makes the logic generally harder to understand, takes up more screen real-estate and is just generally annoying because it needs to be abstracted out by your mind when processing the logic of the decompiled code.
34	the sequence of conditions that have to be checked is clearer
37	Repetition is fine for short instruction sequences else the readability will be affected
38	The structure of snippet 1 is much better in general, because blocks 6, 7 and 11 don't need to be in the loop. If the code duplication is too much, i would prefer the worse structured code, though.
39	Just the length of the code
40	As soon as multiple lines are duplicate i am caught up checking if they are actually duplicate, or if there are small nuances/differences that i might overlook
41	to me its about the visual impact of the block more than its true length or complexity, option 3 has a huge visual impact because it is both long (number of lines) as well as wide (lots of long-ish text literals). option 2 is right at the borderline for me. i'd be okay with either copying or not copying in this case
42	For small snippets I only care about how easy it is to follow and understand the control flow. For larger snippets (5 lines or more?) less repetition means less time spent on trying to understand the same code twice.
43	1. Keep your code DRY, always! 2. Snippet 2 is easier to read because the code is better structured
46	An easy structure makes it much quicker to understand the code. So I always prefer an easiert to read structure
47	Not copying longer snippets feels more natural and likely closer to the original source.
48	Length
50	Code length
51	When decompiled code is repeated, i tend to refer to the disassembly for the memory address for the instructions. If the code is repeated in the decompiled code, it may get quite confusing if they are pointing to the same memory address. For a single instruction such as Option 1, it is still manageable, but for Options 2 and 3, not repeating the block may seem tidier and easier to understand
52	Amount of things to read, only to realise it's a copy Even if realised, it's still visually irritating, and tedious to manually refactor With Snippet 1 it's okay but still convenient to have just one
53	No preference for return instructions. However, i do like decompiled codes to look like proper source code and i feel that most programmers will not have multiple blocks of similar instructions in their source code.
54	If Block 10 contained interesting operations or needed a more in-depth look, having it in one location lets me focus better. Usually I like to comment/highlight, having multiple copies just means that I'd need to copy my highlights and comments to duplicate blocks.
56	The number of lines of code. Prefer larger code blocks to not be copied/repeated as it can make it harder/take longer for users to understand that such code blocks are actually the same.
57	If the instruction sequence in Block #10 is too long, the overall code appears cluttered and becomes more difficult to read, even though it only contains the same instruction sequence multiple times.

58	Unless the block is a one-liner that branches, ("return, break, continue, call"), I prefer to not have a block of code copied out.
60	number of instructions. snippet 2 is preferred if there are many instructions
63	I think that in snippet 2 the general structure of the program and how/ when we reach Block #10 is easier to see.
64	if repetition is avoidable, that's output I would prefer as it allows to code paths to merge into one point eventually. If the repetition is comprised of 1,2,3 instructions, that may be helpful, especially if the code path ends there (return) - means potential code paths do not have to be followed mentally anymore.
78	option 3 is too lengthy
79	I feel that a longer instruction if possible should not be repeated but only once while a shorter instruction can be repeated to simplify the code structure
80	duplicate code can lead to reading the same things twice even though it's doing the same. So if a decompiled decides to duplicate code, it should only do so for portions that one can be understood very fast.
82	Length of code.
84	preference to avoid repeated codes blocks, even if it is 5 lines.
85	More complex the code block should not be duplicated.

Field summary for ConditionIfElseFunc

Now, please assume that /* Block #10 */ contains a more complex structure, for example the following:
Snippet 1 and Snippet 2 are structured as before. In Snippet 3, /* Block #10 */ is extracted as a function.
Snippet 1 Snippet 2 Snippet 3 Which Snippets do you like?
(Multiple choices are possible)

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (SQ001)	1	1.85%
Snippet 2 (Not copying /* Block #10 */) (SQ002)	27	50.00%
Snippet 3 (Extracting /* Block #10 */ as a function and calling it twice) (SQ003)	46	85.19%

Field summary for ConditionIfElseFFree

Please explain your choice.

Answer	Count	Percentage
Answer	49	90.74%
No answer	5	9.26%

ID	Response
1	by declaring a function the multiple calls are easier to read than just including the code multiple times
3	Introducing _inline_ functions complicates the programm which i am not a fan of. In this case, as block#10 containes a return-statement, I feel that it should always be at the bottom, so that the control flow is easier to read. Ghidra often uses the 4th option of using a goto, more or less combining the 'simpler' scoping without copying. But gotos can also be confusing.
6	Minimal duplication of instructions would be great. Abstracting logic to a seaprate function to improve the readability for while loops is welcome too.
7	Snippet 1 creates visuals that cause the code to appear more complex than it must be. My favorite would be Snippet 2, because it provides more "sequential readability". Snippet 3 is fine, too, but would require me to read the code less sequentially. This MAY be more beneficial depending on the actual code block and actual function length, because I can analyze the "outsourced" code more isolated. Afterward, I know what comes out of this function and know directly what's going on when I see the function call.
8	I do like this approach since it simplifies and shortens the information on the first look. I'd also suggest to have distinct function names specifically for the decompiler-generated functions.
10	the reduced complexity in the remaining code due to copying, seems to be balanced with the added complexity of the duplicated block. hiding complexity behind function calls seems much cleaner
12	Not #1, for the reasons I've explained before. #2 and #3 are acceptable, especially if I can label code lines to quickly know where the break ends up. #3 would be preferred if there is some way to indicate that the call is inlined (just to indicate that there is no actual call happening), and if the extracted routine is complex enough and used often enough to warrant extraction. Extracting out too many of these into functions that don't get reused often enough might hinder understanding.
13	Snippet 3 make sit very clear in which case we run that code. Snippet 1 does this also, but bloats the logic in the while loop, causing it to be harder to read. Snippet 2 is still favorable over Snippet 2, but the cases in which the block runs is more implicit and hence also harder to read as Snippet 3.
14	Snippet 2 will probably be more accurate as it doesn't add additional function calls, but snippet 3 will probably be more readable
16	Functions are "clean code" and it's easy to see, that this is a "functional" block, which does exactly one thing and which variables are neccessary for that thing. On the other hand, too much function jumping may hinder the analysis as well, if you get lost and don't know, where you came from.
17	Same as before; code duplication is avoided in snippet 2 and 3. I prefer 3 over 2.
18	It reduces the number of code lines and is very readable with an appropriate function name.
19	Snippet 3 is cleaner and prevent un-necessary re-analysis of code snippets (as seen in snippet 1).
22	Makes things look simpler as long as it is clearly mentioned that it is a decompiler only function
23	Would rather have the block in a single location as explained previously. Also would rather the block not be extracted out as a function as I would prefer seeing everything in one place (rather than clicking on the call_sub function and to see what's going on).
26	Snippet 3 retains the clarity of the while loop without introducing break statements. The copying of the "call_sub" is fine as it is only a single line. Furthermore, user can rename the "call_sub" function to make it descriptive.
29	Extracting it out as a function further helps us in identifying a unique block of code across different sections, and is naturally more helpful for interpretation

30	Helps to meet the criteria I stated for my choices in the previous question; flows better, and helps to shorten the code.
31	It avoids both problems of the while true loop and comparing long snippets. I would prefer the function to be marked "inline", and would not mind a goto-label structure if used once per function.
32	Snippet 3 is able to achieve a simplified structure (ie. easier to understand the control flow compared to snippet 2) but still keep the decompiled code neat and not introduce too many lines of repeated code which takes up more screen space.
34	refactoring is lit
37	Length of instruction sequence is too long for snippet 1. It is fine to be inlined in snippet 2 or as a function call
38	For such a large section, function extraction is a very good choice.
39	Extracting functions is the way to go, at least for anything larger than a few lines in my opinion
40	Now i dont have to worry if the snippets are actually identical, as i described before
41	the block is again on the borderline of visual impact for me, but I dislike excessive break-ing. snippet 3 is a really elegant solution, both minimizing excessive breaking as well as having to re-extract the same semantics from a block of code
42	As above, less repetition means less time spent on trying to understand the same code twice. Extracting the code snippet into its own function yields a very readable control flow but also implies adherence to calling conventions! Depending on the analysis questions this can actually be counterproductive, especially if the analyst blindly assumes that the original assembly code also adheres to a calling convention here.
43	1. Again the DRY argument 2. extracting code to a separate function (separation of concerns) is good for readability
46	It looks like its the most readable version
47	Snippet 2 feels closest to what original source would be and is more 'readable' than Snippet 1. Extracting blocks as functions can get confusing if they reference local variables.
48	More clean
50	If the code block is to be duplicated, I would prefer it to be either short, or made into a function like Snippet 3.
51	Code is more presentable and understandable, compared to snippet 2 where there is a while(true) loop waiting for break conditions. Snippet 3 has a clear exit condition for the while loop.
52	At least it's still one visual block to read
53	Like i said, snippet 3 looks more like a proper source code. Snippet 2 might be more complex but it is still acceptable. Snippet 1 does not look like how a programmer will write his source code.
54	Similar reasons as above, but no real preference between snippet 2 or 3.
56	Both choices allows me to understand more quickly when a code block will be executed. Better preference for snippet 3 if the extracted code block is relatively large, making the code more modular and easier to understand.
57	Even though Block #10 appears in both Snippet 2 and Snippet 3 only once, Snippet 3 appears to be more readable as it's branching seems to be less complex compared to Snippet 2. Also extracting code which is used at multiple locations into a named function makes it easier to remember what the code sequence actually does.
58	Unless the block is a one-liner that branches, ("return, break, continue, call"), I prefer to not have a block of code copied out. Snippet 3 here simplifies the block into a one-liner, which falls into the ("return, break, continue, call") category. In such cases where a block is "extracted" into a function, I think it'll be good to also have a hint that the function does not actually exist.
62	I like snippet 3 the best because it removes the code duplication from snippet 1. Although the function is called twice, it is more readable than snippet 2 because the structure is easier to follow.
63	As above I think not duplicating the snippet makes the program structure easier to see. Extracting it as a function has a similar effect to just using the snippet once.
64	Here, snippet 2 merges potential code paths at a point where something relevant is happening (user interaction). Function extraction (snippet 3) looks just as good to me - as long as it is clearly stated that the output has been optimized for comprehension and is not as close to its disassembly source/structure.
78	cleaner structure and easier to understand. can simply go through the function once to understand the operations involved.
79	I would prefer snippet 2 more because of non repeated codes but snippet 3 is also fine if

	repeat is necessary as making it into a function tidies up the structure.
80	Especially for a very simple case like this (only called ones, instead of `/*logic*/ return var;` it is just `return foo(); int foo() /* logic */`), hat the decompiler should imho not create artificial functions.
82	Fewer lines of code.
84	neater and avoids repeated code blocks
85	The criteria to enter the Block #10 only occurs at 1 place..
86	first snippet increases cognitive load while trying to match both blocks third snippet could distract me from code comprehension in terms of existent functions (which are present in real in binary) and virtual (which introduced by decompiler)

Field summary for CopyMultExit

Please consider the following two code snippets which both represent the same semantics with a different structure. In Snippet 1, /* Block #10 */ is copied to achieve a simplified structure while in Snippet 2 it is not. Snippet 1 Snippet 2 Which structure do you prefer in general?

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	19	35.19%
Snippet 2 (Not copying /* Block #10 */) (A2)	22	40.74%
No preference (A3)	10	18.52%
No answer	3	5.56%

Field summary for ConditionMExitInstr(SQ001)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 1 (1 Instruction)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	40	74.07%
Snippet 2 (Not copying /* Block #10 */) (A2)	7	12.96%
No preference (A3)	6	11.11%
No answer	1	1.85%

Field summary for ConditionMExitInstr(SQ002)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 2 (5 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	17	31.48%
Snippet 2 (Not copying /* Block #10 */) (A2)	27	50.00%
No preference (A3)	9	16.67%
No answer	1	1.85%

Field summary for ConditionMExitInstr(SQ003)

Now, please consider the following possible instruction sequences as replacement for /* Block #10 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #10 */? [Option 3 (15 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (A1)	4	7.41%
Snippet 2 (Not copying /* Block #10 */) (A2)	46	85.19%
No preference (A3)	3	5.56%
No answer	1	1.85%

Field summary for ConditionMExitFree

Please explain your choices. Which criteria guide your decision?

Answer	Count	Percentage
Answer	47	87.04%
No answer	7	12.96%

ID	Response
1	only one return statements helps to understand the cfg better
3	A return statement inside of a while-statement I generally find confusing. In this case the scope-depth seems to be quite low and thus I would avoid copying in all cases. If the scoping depth is really high, or there is a lot of complex code around it, ghidra sometimes introduces a goto in cases like this.
6	Reducing duplication is still preferred, as per the previous question.
7	The code's expressiveness, cumulative length, and block size. The longer the block, the more dangerous it is to overview slight differences when it would be "copied".
8	in this case, it's smarter to just break the while loop.
10	longer sequences that are duplicated add to complexity
12	Similar reasoning as previous question, but now because of additional changes of introducing the temp variable and increased control flow complexity, I feel that I would be more willing to tolerate a few extra lines.
13	To copy one or two lines is preferable for clarity, but more than that can bloat the code and makes it harder to read.
14	I find the additional logic of the exit variable more confusing to follow.
16	If the block gets longer, it is more difficult to see, if it's the same or not. Therefore not copying the block seems more convenient. If depends on the complexity of the overall code though. If the code becomes more difficult to read and more nested, if the block is not copied, the other option may be better.
17	The control flow in snippet 2 is too complex to easily understand.
18	it feels less complicated to read when the code is more segregated.
19	As Previously stated, shorter codes that needs not be re-analyzed can be repeated, and repeats of longer code snippets should not.
22	If its too long then copying will overcomplicate things
23	Easier to analyze control-flow if there is just a simple return statement. However when there are operations done before the return, I would rather have them all in one place (not copying) since it would simplify my analysis. For example, if there are a huge chunk of operations done before the return block, I may have forgotten what was being performed in the return and attempt to analyze the return block again. I rather have them in one place so I can add comments on their meaning in a single location where I can refer to.
26	Copying is fine for short instruction sequences. Not copying makes sense for long instruction sequence. But the multiple exits in the while loop makes it harder to read.
29	Same as earlier
30	Same as previously, I like having the flow in Snippet 1, and the "exit" flag and immediate check feels strange to me. But I can see the argument for shortening it if the code blocks are too long.
31	- A while-true loop with many breaks and continues is confusing, so I prefer the while-condition loop with the copied block. - It is hard to compare longer snippets and notice that they are identical.
32	I can see the value of simplifying the structure like in Snippet 1 but in this case, the control flow is easy enough to understand and IMO is not worth the trade-off of having to repeat the same code block.
34	isn't this the same thing as the last part
37	Option 1 is short enough to be repeated but not option 2 and 3
38	As in the last step, I would prefer duplication as long as its not too much. I like function function extraction even more, though. P.S.: What happened to Block #3 in the first example?

39	Readability mostly
40	Same reason as in the last question (hard to see if actually identical or different)
41	i dislike excessive break-ing and the branch conditions are more complex in snippet 2, but the visual impact of option 3 is too large for copying
42	Less repetition means less time spent on trying to understand the same code twice. But if I have to pay for it via more complex control flow conditions (like that exit_1 variable), I lean on preferring easier control flow most of the time.
43	If block 10 would be moved to a separate function, I would always prefer snippet 1
46	I prefer an easy structure instead of less code
47	Following control flow in Snippet 2 is difficult and only tolerable in Snippet 1 for the simple return statement.
48	Length
50	Code length. Prefer to have less lines of code if the code block is going to be duplicated.
51	Multiple exit conditions with break seems more confusing compared to a while loop with a fixed exit condition. But if the repeated block gets too large, not repeating it seems more reasonable
52	IMO Option 1 and Option 2 is not worth the effort of having to follow Snippet 2?
53	I prefer the simplicity of snippet 2. It is also more understandable.
54	Similar explaination as before - Not Copying makes it easier for analysis.
56	For larger code blocks that are copied, it takes more time to realize that both blocks are actually identical and hinders program understanding.
57	If the instruction sequence is relatively short, copying the sequence into multiple locations appears to be a good trade-off between overall code length and a more complex. The instruction sequence in Option 2 seems to be a good boundary, whereas the seuquence in Option 3 would make Snippet 1 appear more cluttered.
58	Similar to the section before.
62	Option 3 is the codes, which shows most explicit what exactly it does. Although it is more lengthy than the other options, it is the code I best understand / can read. Because, option3 would be for snippet 2 called outside the while statement, it is best, if the intention of the instructions are clear, without the context of the loop. I chose option1 for the first snipped, because of its simplicity. Especially if the code is called twice and is not wrapped by a function, it is preferable if the code is then simple.
63	When not copying the snippet, the structure is easier to see.
64	I think I already had this question? Anyhow, short code copied (1,2,3 instructions) is fine, for longer instruction sequences, merging flow is preferred.
78	too lengthy
79	Would prefer codes not to be repeated but if necessary then the shorter instructions could be repeated
80	same as before
84	preference for not reading code blocks that are identical.
85	complex code should only be evaluated once or minimally.

Field summary for ConditionMExitFunc

Now, please assume that /* Block #10 */ contains a more complex structure, like e.g. the following:
Snippet 1 and Snippet 2 are structured as before. In Snippet 3, /* Block #10 */ is extracted as a function.
Snippet 1 Snippet 2 Snippet 3 Which Snippets do you like?
(Multiple choices are possible)

Answer	Count	Percentage
Snippet 1 (Copying /* Block #10 */) (SQ001)	4	7.41%
Snippet 2 (Not copying /* Block #10 */) (SQ002)	26	48.15%
Snippet 3 (Extracting /* Block #10 */ as a function and calling it twice) (SQ003)	47	87.04%

Field summary for ConditionMExitFFree

Please explain your choice.

Answer	Count	Percentage
Answer	48	88.89%
No answer	6	11.11%

ID	Response
1	declaring the duplicate code in a function looks cleaner. multiple return statements require an extra look
3	Introducing a function to me seems like introducing complexity, which seems bad, but as ghidra does not do that I don't really now how it would work out. It might be worth a test. In this case (the same reason as above) I would vote to not copy the block.
6	Reducing duplication is still preferred, as per the previous question.
7	Again: It would be too convenient for my brain to see the redundancies of a complex copied block. I'd scan the code real quick and then just skip it. However, what if there are slight nuances in code that differ? That would be too dangerous for me.
8	When encountering somewhat big repeated code-block, I'd prefer extracting the block as a function.
10	complexity between 1 and 2 seems to be balanced. snippet 3 looks clean
12	Same reasoning as before, but here I am leaning more towards call_sub because of the increased control flow complexity in #2.
13	The function call in Snippet 3 makes it very clear when the block is executed. In Snippet 2, this is more implicit and hence harder to read. Snippet 1 makes this very clear, but bloats the code logic, causing it also to be harder to read as Snippet 3.
14	Same as above, I find the additional logic of the exit variable more confusing to follow. And same as previous question, extracting as a function can potentially make it more human-readable but may not be the truest representation of the program.
16	It seems more clean, like "clean code" and it is easy to see, that the function is a block on its own, which does one thing and to see, which variables are neccessary for it. To much function jumping on the other hand may not be beneficial too, because it's easy to loose track, where you came from.
17	Snippet 3 is easy to understand with a simple control flow.
18	substituting function calls with makes it easier to read.
19	Less analysis required and no repeated functions. Functions are read 1-pass and in the case of snippet 3, code is separated in different function and can be analyzed seperately
22	Looks similar to how you would do in C code
23	Would prefer the return block in one place as explained previously. Also would prefer the block not be extracted out as a function as I would rather have everything in one place for easier analysis (I don't have to click the call_sub function to analyze it).
26	While loop is more readable without multiple exits. User can also rename the "call_sub" to make it more descriptive.
29	Same as earlier
30	Like the previous question, I like to extract as function to allow me to maintain the flow while reducing code length.
31	It avoids both problems of the while-true loop and comparing long snippets. I would prefer the function to be marked "inline", and would not mind a goto-label structure if used once per function.
32	Repeated code blocks is quite annoying and requires more focus to abstract out when understanding the decompiled code logic.
34	refactoring is still lit
37	Snippet 2 and 3 are cleaner and easier to read
38	I think function extraction makes sense here (because of the length of the block)
39	Refactor anything that is longer than a few lines and occurs multiple times
40	again same reason as in the last question (easier to see that its identical)

41	again, excessive breaking in snippet 2 and more complex branch conditions visual impact of the block is near the borderline for copying
42	As mentioned in a previous question, I do not like the implied adherence to a calling convention in snippet 3 if it may not be true.
43	Has the advantage of having the simpler/more readable structure with benefit of DRY code
46	just the easiest to read and understand
47	Removing gotos in this case causes nothing but pain
48	More clean
50	Prefer not to have duplicated code blocks if there are many lines of code. If there is, would be good to have it in a function like Snippet 3.
51	Code looks easier to understand with while loop exit condition clearly defined
52	IMO feels the cleanest and easiest to follow?
53	Each of the functions are smaller and much easier to understand.
54	As above.
56	For more complicated blocks I prefer if it is not copied again or it is extracted into a function for quicker understanding of the code.
57	Extracting Block #10 into a separate function as in Snippet 3 is the most favorable solution as it enables to decrease the overall code length and the complexity of the code structure. Snippet 2's structure is more complex than the structure of Snippet 1, but as Block #10 contains a more complex structure, copying the code block into multiple locations would increase the overall complexity of Snippet 1.
58	Similar to the section before. Note that the "extracted" function should have a hint that it is a real function.
62	Snippet 1 is the snippet which I like the least, because of the code duplication. I prefer the structure of snippet 2 over the wrapping function of snippet 3 because its instructions are less difficult to understand for me.
63	When not copying the snippet, the structure is easier to see. Replacing the snippet with a function call also makes this easier, although having the snippet only once is still the easiest.
64	same arguments as before, relevant functionality happening after code flow is merged is nice and function extraction is fine as well.
78	easier to follow as the code is structured
79	Snippet 2 & 3 makes the code looks tidier as there are no repeats of code compared to snippet 1.
80	Snippet 1 duplicates a lot of code, so absolutely not an option. Snipped 2 needs to use the helper variables to avoid the duplication, which adds cognitive load, hence Snippet 3 is preferable.
82	Fewer line of code.
84	neater and easier to understand.
85	Criteria to enter into Block #10 is only evaluated once.

Field summary for CopyMultEntry

Please consider the following two code snippets which both represent the same semantics with a different structure. In Snippet 1, /* Block #4 */ is copied to achieve a simplified structure, while in Snippet 2 it occurs only once. Snippet 1 Snippet 2 Which structure do you prefer in general?

Answer	Count	Percentage
Snippet 1 (Copying /* Block #4 */) (A1)	27	50.00%
Snippet 2 (Not copying /* Block #4 */) (A2)	17	31.48%
No preference (A3)	9	16.67%
No answer	1	1.85%

Field summary for ConditionMEntryInstr(SQ001)

Now please consider the following possible instruction sequences as replacement for /* Block #4 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #4 */? [Option 1 (1 Instruction)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #4 */) (A1)	47	87.04%
Snippet 2 (Not copying /* Block #4 */) (A2)	1	1.85%
No preference (A3)	5	9.26%
No answer	1	1.85%

Field summary for ConditionMEntryInstr(SQ002)

Now please consider the following possible instruction sequences as replacement for /* Block #4 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #4 */? [Option 2 (5 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #4 */) (A1)	22	40.74%
Snippet 2 (Not copying /* Block #4 */) (A2)	21	38.89%
No preference (A3)	10	18.52%
No answer	1	1.85%

Field summary for ConditionMEntryInstr(SQ003)

Now please consider the following possible instruction sequences as replacement for /* Block #4 */:
Option 1 Option 2 Option 3 Which structure do you prefer depending on the length of the instruction sequence in /* Block #4 */? [Option 3 (15 Instructions)]

Answer	Count	Percentage
Snippet 1 (Copying /* Block #4 */) (A1)	9	16.67%
Snippet 2 (Not copying /* Block #4 */) (A2)	39	72.22%
No preference (A3)	3	5.56%
No answer	3	5.56%

Field summary for ConditionMEntryFree

Please explain your choices. Which criteria guide your decision?

Answer	Count	Percentage
Answer	43	79.63%
No answer	11	20.37%

ID	Response
1	it reduces the while loop to one block which helps understanding it faster
3	This example is a little bit weird. As Block #4 returns snipped 2 could be drastically simplified. Further, I can't really tell what the underlying machine code is. If there is an actual variable 'entry_1' (either stack or register) then eliminating the code that sets and reads the value, is not acceptable. But probably the 'if(var_0 > 10){}' just jumps into the while, in which case the c-scoping semantics are not really obeyed and the cleanest solution would to be a goto.
6	While it might be personally more cumbersome to read, the use of entry flags can help to signpost possible code flows.
8	in this case, I'm hesitant to choose the copy feature because sometimes the logic is repeated for a reason, but here you would make it less obvious.
10	breaking condition in snippet 2 is less complex than sequences in option 2 and 3 that would be duplicated
12	Similar to previous question, here I think I would lean more towards copying for similar reasons as before.
13	Because the nesting is so shallow, copying short blocks is preferable for clarity in this case.
14	Same as previously, I find the additional logic of the entry variable more confusing to follow.
16	Since the code gets more complicated, if not copyied, but is very easy, if copied, option 1 and 2 are more convenient for snippet 1. But in option 3 the block is very large and snippet 2 becomes more convenient.
17	Only entering a loop to break it in the first iteration can be hard to follow, so I would prefer to avoid snippet 2.
18	Snippet 1 has a significantly more simplified structure to tolerate the copying.
19	single line and small (simple) code snippets can be repeated, but long snippets should not be to prevent unnecessary re-analysis of code.
22	Length of copied code
23	In snippet 2, I would only have to analyze the if statement in line 10 to determine if the block will be executed at least once. However for snippet 1, I have to analyse the if statement in line 4 and the while statement in line 9 to determine if the block will be executed.
26	Length of the copy affects readability.
29	Same as earlier
30	Same as previous.
31	- I prefer while-condition loops to while-true loops - The entry_1 flag is hard to keep track of - It is hard to compare longer snippets and notice that they are identical, so I prefer not copying them.
32	Copying code here does make the conditions for block #4 to be executed clearer. However, if block #4 is large, it is going to be annoying to read.
34	while(true) is annoying with the exit conditions
37	Shorter instruction sequence for readability. The logic flow is not lost in snippet 2.
38	The structure in Snippet 2 is really ugly and hard to unwind (when the actual blocks are filled in).
40	the "copying" version is for more readable for me, but if there are too many instructions it becomes messy again
41	while the visual impact of option 3 is significant, in this case the increase in complexity of snippet 2 compared to snippet 1 outweighs the visual impact
42	Side note: All options end in return instruction, which seems somewhat wrong considering the code of the two snippets. Except for repetition of large snippets I prefer less variables that need to be followed to understand the control flow.

47	I dislike the use of extra variables to denote control flow in Snippet 2, and copying large blocks is undesirable in Snippet 1. while() {return} is also terrible decompilation
48	Length
50	Code length for Option 1. Option 3 is too long with many function calls.
51	If there are multiple entry within loops, would prefer snippet 1 to show the possible entries into the block. However, if the block is huge, I would not prefer either snippets, as the block should be modularized into a function on its own.
52	Option 1 and Option 2: IMO copying is easier than having to follow the entry variable? Not sure for Option 3
53	Simplicity, ease of understanding
54	Same reasoning as before. Less duplicated code means less bloat.
56	Size of code block. For larger/more complicated blocks I would prefer for it to not be repeated as it makes it harder to realize two code blocks at different parts of the code are actually the same.
57	This time the structure of Snippet 1 and Snippet 2 appear to be equally complex. Thus, copying Option 2 could be avoided as it does necessarily bring a less complex structure. Option 3 appears to be too large to be copied into multiple locations compared to the reduced structure complexity that is achieved.
58	Similar to the section before.
62	I choose option 1 for snippet 1 to reduce the code duplication. Option 3 is in my opinion better suitable for snippet 2 because the code is more 'expressive', meaning it shows in a meaningful, explicit way what is exactly does, whereas the simple return statement might be unclear, what exactly being returned.
63	The criteria in line 10 and 11 of the second snippet were a bit hard to read, but for a longer block 4, this would be worth not duplicating the long code block.
64	overall, I like snippet one more because it has simpler conditional expressions. but once again, if the copied code portion becomes too long, I'd rather have it once during a time of mostly non-conditional, linear code flow.
78	length of the block
79	As the codes in snippet 1 is shorter than snippet 2, option 1 being repeated is fine but if the instructions are long, then snippet 2 will be easier to read
80	same as before
84	for 1-liners, it is ok to duplicate the instructions. otherwise, it is preferred for identical code blocks (functionality) to be called once.
85	Complex code should be evaluated once or minimally.

Field summary for ConditionMEntryFunc

Now please assume that /* Block #4 */ contains a more complex structure, like the following: Snippet 1 and Snippet 2 are structured as before. In Snippet 3, /* Block #4 */ is extracted as a function.
Snippet 1 Snippet 2 Snippet 3 Which Snippets do you like?
(Multiple choices are possible)

Answer	Count	Percentage
Snippet 1 (Copying /* Block #4 */) (SQ001)	8	14.81%
Snippet 2 (Not copying /* Block #4 */) (SQ002)	21	38.89%
Snippet 3 (Extracting /* Block #4 */ as a function and calling it twice) (SQ003)	48	88.89%

Field summary for ConditionMEntryFFree

Please explain your choice.

Answer	Count	Percentage
Answer	42	77.78%
No answer	12	22.22%

ID	Response
1	function identifies the cide block as a consistent piece that can be understood on its own without the need of identifying duplicate code
6	Minimal analysis of potentially long code still takes priority.
8	the same explanation as before
10	again: easy breaking condition with complex duplicated code. having code duplication inside function looks nice
12	Similar reasoning as before, but it is a closer choice that I might pick 1.
13	The function call makes it very clear when the block is executed and don't obstruct the logic. Snippet 1 also achieve this, but is more bloated than Snippet 3. Snippet 2 does this more implicit, but is still harder to read.
14	Same as previously, I find the additional logic of the entry variable more confusing to follow. And same as previous questions, extracting as a function can potentially make it more human-readable but may not be the truest representation of the program.
16	Since the examples are not that complex and long, all 3 options seem to be fine.
17	Snippet 3 is the best solution due to the simple control flow.
18	substitute with functions calls makes it easier to read
19	single line and small (simple) code snippets can be repeated, but long snippets should not be to prevent unnecessary re-analysis of code. Having the code separated out as a function also allowed for cleaner analysis as the function can be analysed as a separate unit
22	easiest to understand
23	Same reasoning as the previous part (would prefer not extracting the block as a function)
26	It's more concise to have everything viewable from a single function. Furthermore, it doesn't create any artificial function calls. Perhaps it would be useful to allow user to collapse the while statement.
29	Same as earlier
30	Same as previous
31	It avoids all the problems of the while-true loop, flags, and comparing long snippets. I would prefer the function to be marked "inline", and would not mind a goto-label structure if used once per function.
32	As mentioned earlier, having to read repeated code blocks in 1 function (Snippet 1) is annoying.
34	once again, refactoring is lit
37	Easier to understand a more complex set of instructions when they are extracted out into a separate function
38	Function extraction is the best choice here, IMHO.
40	I find the copying version easier to read, and now there is no duplicate code.
41	snippet 3 is by far my favourite but I'd be okay with analyzing snippet 1
42	Snippet is not long enough for me to mind the repetition. I would actually prefer snippet 3 if it were not for the implied adherence to a calling convention that may not actually be true.
47	Like in the previous section, removing gotos leads to worse decompilation results here.
48	Clean
50	Snippet 3 will be very clear if the decompiler supports clicking of call_sub to go to the decompiled code of the call_sub function.
51	Snippet 3 is the clearest and easiest to understand. The repeated block of code is modularized into a function where multiple entries to the block are labelled function calls.
52	Easiest to follow
53	It looks like a proper source code. Plus, individual functions are smaller which are easier to analyze.
54	As above.

56	Prefer larger/complicated code blocks to not be repeated across the code for quicker understanding of program behaviour.
57	Extracting the code into a separate function still is the most readable and least complex solution. However, as the overall complexity of Snippet 2 is somewhat equal to the complexity of Snippet 1, Snippet 2 seems also OK.
58	Similar to the section before.
62	I chose snippet 3 because it has the easiest structure without any code duplication. Hence, it is easy to read.
63	Here in snippet 3 makes getting an overview about the functionality of the code the easiest.
64	given my previous arguments, snippet 3 is best of both worlds.
78	shorter code and easy to follow structure
79	Snippet 2 & 3 looks cleaner & readable than snippet 1
80	same as before
84	neater and easier to read.
85	The conditions leading to Block#4 is processed prior to the call.

Field summary for SwitchCases2Snippets(SQ001)

Please consider the following two code snippets. Snippet 1 Snippet 2
[Which snippet do you prefer?]

Answer	Count	Percentage
Snippet 1 (A1)	24	44.44%
Snippet 2 (A2)	15	27.78%
No preference (A3)	15	27.78%
No answer	0	0.00%

Field summary for Switch2Free

Please explain your choice.

Answer	Count	Percentage
Answer	47	87.04%
No answer	7	12.96%

ID	Response
1	for snippet 2 it's not clear whether it's possible to enter multiple if conditions
3	I assume the compiler did not emit a jump-table and in this case, as there are not even any 'else'-statement it is likely that Snippet 2 is more accurate and has less scoping.
6	Personal preference (even from a software engineering perspective). Less Indentation too.
7	snippet 2 reduces one depth of indentation. Feels more natural for me to read.
8	simple condition blocks are rather lengthy when implemented as switch-case block
10	snippet 2 is shorter and clear to read
12	There are not enough cases for me to consider a switch an upgrade yet.
13	I think the logic in a switch statement is more obvious and nicer to read than if-else blocks.
14	Both are still quite readable.
16	Just don't like switch statements.
17	Snippet 1 highlights the exclusiveness of the two arms which is not immediately visible in snippet 2 due to the missing "else if". Also, it's easier to spot that we are comparing the same variable against multiple constants in snippet 1.
18	nil
19	The readability of Switch Cases and Ifs are dependent on the code snippet prior, and which determines the flow of the code. The readability of switch cases/if statements can drastically change based on the value that undergoing the logic statements.
22	switch statement seems unnecessary. Also not sure if they have the same meaning as snippet 2 can possibly execute both blocks 1 and 2 but snipper 1 cannot.
23	If statements are more intuitive to me if there are only a few cases.
26	If statement is more concise when it's just 2 cases.
29	Case switch presents a more intuitive understanding of the flow of the code.
30	Two choices I don't think calls for a switch, just a personal preference. I prefer when switch is used for numerous options, but for two I prefer simple ifs.
31	Both are fine, as they are short and uniform. - The switch-case structure has no weird fall-throughs - The if conditions have no overlaps
32	neater, less braces :) and if it's a long statement, you'll immediately know that the control flow is determined by 1 variable (var).
34	I immediately see that I only need to think of the value of var for all of this switch statement
37	Depends on the length of the switch case/if-else. Switch case might be more appropriate for a greater number of conditions while multiple if-else has better readability when representing a small number of conditional statements
38	When there are no nested `else if ()` constructs and only two options a switch isn't really worth the effort
39	For just two cases, if-elif is better in my opinion.
40	switch cases make it easy to directly see the structure of the code
41	the switch statement adds inherent semantics compared to the if conditions, making it easier to glance the purpose of the code
42	Snippet 1 makes it clearer that the code cannot execute both block 1 and 2 but only one of them.
43	Both snippets are easy to read. Anything more complex than that should be a switch
47	Both appear to be plausible, but with only 2 possibilities, Snippet 2 is readable enough.
48	Easier to read
50	Easier to understand if there is not too much nested if-else statements.
51	Both snippets are straight forward and easy to understand.
52	Easier to read and list the important values of var?
53	No preference. They look the same to me.

54	I find switch statements slightly easier to read here, but I'm fine with either.
56	Easier to determine that the case statements depend on the same variable, especially when there are a lot of conditional (case/if-else) statements.
57	The Switch-Statement and the If-Statements appear to be equally complex. Thus, no real preference.
58	No preference if the number of differentiating conditions (2 in this case) are small.
62	Although switch case statements might be a little lengthier than the if statements from snippet 2, they have the better and cleaner structure. In specific, any case refers to the same variable, whereas in the multiple if statements this can be potentially mixed up and therefore difficult to read. Furthermore, the switch statements encapsulate all cases so that it's obvious, when the parsing ends.
63	Here both are equally readable and the if construction is more familiar. But in the first example in the survey with the multiple nesting layers for the (else) if structures, these became quite confusing and the switch case construction would be preferable.
64	I'd say this depends on the length of block #1 and #2, which is not shown. Otherwise I probably like snippet 1 more because the switch can be understood as one big coherent block, while I would have to check in snippet 2 if var is modified in a way that would lead to execution of block #2.
78	similar in length and both snippets are still easy to comprehend
79	Looks cleaner & condition only validate once
80	switch cases are a great way to express the connected-ness of branches, hence I prefer it.
84	if..else for switch case is easily readable, no difference.
85	Switch statement contains less visual artefacts (it's neater)
86	in short switch case while `switch(var)` is visible it's good to have snippet one

Field summary for SwitchCases5Snippets(SQ001)

Please consider the following two code snippets. Snippet 1 Snippet 2 [Which snippet do you prefer?]

Answer	Count	Percentage
Snippet 1 (A1)	37	68.52%
Snippet 2 (A2)	6	11.11%
No preference (A3)	11	20.37%
No answer	0	0.00%

Field summary for Switch5Free

Please explain your choice.

Answer	Count	Percentage
Answer	46	85.19%
No answer	8	14.81%

ID	Response
1	for snippet 2 it's not clear whether it's possible to enter multiple if conditions
3	Same reason as above.
6	Personal preference (even from a software engineering perspective). Less Indentation too.
7	the more constant cases for a single var, the more natural it feels to me to use and, thus, read, a switch case
8	same as above
10	snippet 2 does not look bad, since conditions are easy and we have no nesting. snipped 1 makes it clear that we are checking for var. no preference
12	Here there are enough cases that a switch is more readable.
13	I think the logic in a switch statement is more obvious and nicer to read than if-else blocks.
14	Both are still quite readable.
16	Just don't like long switch statements with blocks in it.
17	Snippet 1 highlights the exclusiveness of the two arms which is not immediately visible in snippet 2 due to the missing "else if". Also, it's easier to spot that we are comparing the same variable against multiple constants in snippet 1.
18	nil
19	The readability of Switch Cases and Ifs are dependent on the code snippet prior, and which determines the flow of the code. The readability of switch cases/if statements can drastically change based on the value that undergoing the logic statements.
22	Looks cleaner
23	If there are many cases, switch is easier to understand. I would know that a block of operations will be performed based on the value of var in one glance.
26	Switch is easier to read when there are many cases.
29	Case switch presents a more intuitive understanding of the flow of the code.
30	I prefer when switch is used for numerous options: for 5 i am okay with both versions, personal preference.
31	Both are fine, as they are still uniform. - The switch-case structure has no weird fall-throughs - The if conditions have no overlaps
32	neater, less braces :) and if it's a long statement, you'll immediately know that the control flow is determined by 1 variable (var).
34	I immediately see that I only need to think of the value of var for all of this switch statement
37	switch case for a bigger number of conditional statements is easier to read and understand
38	because of the sheer number of cases, a switch statement improves readability here IMHO
40	switch cases make it easy to directly see the structure of the code
41	same as above, the switch statement adds inherent semantics compared to the if conditions, making it easier to glance the purpose of the code
42	Snippet 1 makes it clearer that only one of the blocks get executed.
43	Since every condition uses var, it only makes sense to use a switch
47	While Snippet 1 is probably closer to original source code, Snippet 2 is less complicated, so I don't have a strong preference either way
48	Easier to read
50	Even though it is longer, it is still easy to understand.
51	With more if conditions regarding the same variable, a switch-case statement is more readable and neater.
52	Easier to read and list the important values of var?
53	No preference. They look the same to me.
54	As above,switch statements slightly easier to read here, but I'm fine with either.
56	Easier to determine that the case statements depend on the same variable, especially when

	there are a lot of conditional (case/if-else) statements.
57	Switch-Statement and If-Statements contain more cases, but the overall structure seems to be equally complex.
58	For larger number of conditions, a switch-block makes the code easier to read over a series of if-blocks.
62	Same reason as above: Although switch case statements might be a little lengthier than the if statements from snippet 2, they have the better and cleaner structure. In specific, any case refers to the same variable, whereas in the multiple if statements this can be potentially mixed up and therefore difficult to read. Furthermore, the switch statements encapsulate all cases so that it's obvious, when the parsing ends.
63	Again, both are almost equally readable, but the switch case construction shows more clearly that all of the following cases/ if statements will be dependent on the value of this one variable, which gives an overview over the code functionality more quickly.
64	for the same reason that the break statements in snippet 1 suggest a single case being taken at most, this is easier to follow than snippet 2 which may have code in the blocks that may lead to execution of more than one if-body.
78	similar in length and both snippets are still easy to comprehend
79	Looks cleaner & condition only validate once
80	same as above
84	if..else for switch case is easily readable, no difference.
85	Switch statement contains less visual artefacts (it's neater)
86	in case when lot of blocks it's good to see `var == x`

Field summary for SwitchCase2dSnippets(SQ001)

Please consider the following three code snippets. Snippet 1 Snippet 2 Snippet 3
[Which snippet do you prefer?]

Answer	Count	Percentage
Snippet 1 (A1)	46	85.19%
Snippet 2 (A2)	3	5.56%
Snippet 3 (A3)	0	0.00%
Both if-Snippets (A4)	2	3.70%
No preference (A5)	2	3.70%
No answer	1	1.85%

Field summary for Switch2DFree

Please explain your choice.

Answer	Count	Percentage
Answer	45	83.33%
No answer	9	16.67%

ID	Response
1	switch case is easy to read and default condition remains the same during growth of switch cases
3	If Snippet 2 accuratly represents the underlying code, meaning there are no else jumps and in the end it tests for all other conditions being wrong, then Snippet 2 is the best. Otherwise, if the code is an if-else-chain or a jump table, I would prefer Snippet 1.
6	Personal preference (even from a software engineering perspective). Less Indentation too.
7	I think a "default" case is the more expressive choice here (snippet 1). The two-part condition of snippet 2 takes more "horizontal-visual-eye-parsing" and active reading to come to the conclusion that it is basically the default case. Snippet 3 has too much nesting.
8	here it's a special case, it leads to less indentation
10	snippet 1 beats both "complex" condition and nesting
12	I think the switch is easier to understand as there is less nesting and clearer conditions, but it is close due to small number of cases.
13	The switch statement is nicer to read.
14	Snippet 1 is the easiest to follow, snippet 2 has additional logic to process, snippet 3 has too much nesting.
16	Same. Snippet 3 contains unnecessary nesting which complicates reading it.
17	Snippet 1 highlights the exclusiveness of the two arms which is not immediately visible in snippet 2 due to the missing "else if". Also, it's easier to spot that we are comparing the same variable against multiple constants in snippet 1. In this case, the switch also has the simplest control flow.
18	switch-case is most readable out of the 3 choices
19	Switch case looks the cleanest and directly readable in this snippet
22	Looks cleaner
23	In snippet 1, have to perform the additional step of analyzing what the default case mean (var != 1 && var !=2). This means I will comment it at line 9 anyway. Thus I would prefer having the condition explicitly written out in snippet 2 like in line 8. Snippet 3 seems unnecessarily convoluted.
26	Switch is easier to read.
29	Case switch presents a more intuitive understanding of the flow of the code.
30	The default case for switch looks cleanest and most straight forward to me, the ifs just complicate things.
31	The final none-of-the-above case corresponds well to the "default" part in switch-case. Other than Snippet 1, I would also be fine with Snippet 3, as the else case rules out overlapping conditions, unlike in Snippet 2 where I need to double-check.
32	neater, less braces :) and if it's a long statement, you'll immediately know that the control flow is determined by 1 variable (var).
34	I immediately see that I only need to think of the value of var for all of this switch statement (why no if - else if - else?)
37	if-else better for 3 conditional blocks
38	Snippet 2 is just ugly and snippet 3 is unnecessarily convoluted with its nested if statements. Furthermore, a default block in a switch statements is very intuitive and therefore easy to comprehend
40	switch cases make it easy to directly see the structure of the code
41	same as above, plus defaults are easier to understand with the switch statement
42	I generally like the readability of switch cases. Snippet 2 is the worst of the three, since one first has to check that block 1 and 2 do not change the variable before one knows that the code is equivalent to the other two. This

	means more analysis work compared to snippets 1 and 3.
43	Intuitively understandable code
47	This is clearly a switch variable and Snippet 1 best reflects that.
48	Easiest to understand the conditions
50	If there are more conditions in the if statements, I prefer the switch-case's default block.
51	Snippet 1 looks neater and easier to understand. Snippet 2 looks too confusing with the last condition, especially when dealing with more conditional statements. Likewise for Snippet 3, the nested conditions will look horrible with many conditional statements.
52	Easier to read and list the important values of var?
53	Snippet 2 and 3 does not look right. Another alternative i would prefer is if (var ==1) else if (var == 2) - else()
54	As above.
56	I find it easier to understand the conditions under which a code block will be executed when placed within switch-case statements. Having a default case makes it clearer that a specific code block is executed when all other conditions are false.
57	Snippet 3 contains nested branching, which seems to have the most complex structure compared to the other Snippets. Snippet 2 shows the same level of branching as Snippet 2. Actually, there is no real preference between Snippet 1 and Snippet 2, but I stick to Snippet 1 as it handles the "default" case in a more elegant way.
58	In this case, the switch-block makes the decompiled code easier to read. There is no need to run through individual if-statements only to know that Block #3 is the "default" case.
62	I still prefer snippet 1, because of its structure. Although snippet 3 has a better structure than snippet 2, it might be potentially difficult to read because it gets more and more nested for each additional IF-ELSE block.
63	I don't think the nested if/ else statements are easy to read. The second snippet is also okay, but line 8 would most likely become less readable for more previous cases7 if statements, so snippet 1 is easiest to read.
64	snippet 1 with the same arguments as above (easy to follow thanks to the breaks) and otherwise snippet 3 over snippet 2 if the nesting remains on this level.
78	easier to follow
79	Looks cleaner & condition only validate once
80	same as above
84	all snippets are still readable. no preference so long as the conditional statements are straight forward .
85	It is more succinct.

Field summary for SwitchCase5dSnippets(SQ001)

Please consider the following three code snippets. Snippet 1 Snippet 2 Snippet 3
[Which snippet do you prefer?]

Answer	Count	Percentage
Snippet 1 (A1)	51	94.44%
Snippet 2 (A2)	2	3.70%
Snippet 3 (A3)	0	0.00%
Both if-Snippets (A4)	0	0.00%
No preference (A5)	1	1.85%
No answer	0	0.00%

Field summary for Switch5DFree

Please explain your choice.

Answer	Count	Percentage
Answer	45	83.33%
No answer	9	16.67%

ID	Response
1	switch case doesn't need to be nested and the default condition remains slim
3	Same reason as above.
6	Both snippet 1 and 2 are fine since the indentation level is relatively significant for snippet 3 here.
7	Snippet 3 is bad because of the high nesting Snippet 2 is bad because the multi-part conditional is way too canonical and complicated.
8	same as above + less logic inside the last if statement
10	eating complex conditions and nesting up in a default case looks clean
12	Much easier to read the switch statement, handles the larger number of cases easily.
13	I still prefer the switch statement. It contains fewer brackets, and it is very clear in which case what happens. Snippet 3 is very complex due to the nesting, and Snippet 2 is better suited for a switch statement.
14	Snippet 1 is the easiest to follow, snippet 2 has additional logic to process, snippet 3 has too much nesting.
16	Same. The default state is nice though, but could be achieved with if/else if/else which would be more logical.
17	Snippet 1 highlights the exclusiveness of the two arms which is not immediately visible in snippet 2 due to the missing "else if". Also, it's easier to spot that we are comparing the same variable against multiple constants in snippet 1. In this case, the switch also has the simplest control flow.
18	switch-case is most readable out of the 3 choices
19	Switch case provides the most readable output for this snippet.
22	Looks cleaner
23	Would prefer snippet 1 over snippet 2 as the last condition (line 17) seems too long for it to be intuitive.
26	Switch is more concise. Nested if statements makes it hard to read.
29	Same as earlier, even more so for this case. It simplifies the structure massively
30	Again, with default cases, switch should probably be used
31	The final none-of-the-above case corresponds well to the "default" part in switch-case. Snippet 2 has a messy final case with too many conditions to eyeball. Snippet 3 has too much nesting, but it would work fine if the formatting allowed "else if" to be on the same line and not indented.
32	neater, less braces :) and if it's a long statement, you'll immediately know that the control flow is determined by 1 variable (var).
34	I immediately see that I only need to think of the value of var for all of this switch statement
37	Easier to understand
38	Is there even a choice here? Snippets 2 and 3 are gruesome in comparison.
40	switch cases make it easy to directly see the structure of the code
41	same as above, still in favour of the switch, only even worse if snippets than before
42	I still like the readability of switch-cases. Here snippet 3 is the worst since it reaches a nesting depth that greatly hurts readability.
43	Intuitively understandable code
47	As in the previous example
48	Easiest to understand the conditions
50	Same reasoning as above.
51	Snippet 3's nested conditions looks like an eyesore, especially when the blocks of code inside are huge. I will not be able to tell the levels of indentation where my block belongs. Snippet 2's last condition is a mess, and very confusing.

	Snippet 1 is the easiest to understand
52	Easier to read and list the important values of var?
53	Same as my previous explanation
54	As above.
56	I find it easier to understand the conditions under which a code block will be executed when placed within switch-case statements. Having a default case makes it clearer that a specific code block is executed when all other conditions are false. It also avoids too much nesting when there are many possible cases.
57	Snippet 3 contains even more nested branches. The expression to handle the "default" case in Snippet 2 is too long. The Switch-Statement appears as it has been made for this kind of branching.
58	I'm not a fan of nested if-blocks. Switch-blocks are much easier to read in this case, given that the matching condition is a single variable. However, in the event where the if-condition has multiple variables, "if (a && b || c)" etc., then it will be difficult to represent the code in switch-blocks. In such cases, if-blocks would be preferred. As mentioned earlier as well, the switch-block is preferred here as it is easy to identify the "default" case.
62	As already explained above, I like the switch statement the most. Although snippet 2 is less nested and a bit shorter, snippet 1 is better to read. Especially the “default” case is clearer than the long IF-statement from snippet 2 at the end (line 17 / 18)
63	The nested if else statements are really hard to read, especially if there are a lot of them and one has to work on a smaller screen, and line 17 in the second snippet becomes too long. Snippet one is the easiest and fastest to read.
64	switch is preferred in this structure. snippet 2 is at least more flat and indentation is really painful once those single-line "block #n" have several instructions.
78	the other two snippets have too much conditions and if-else statements
79	Looks cleaner, condition only validate once & code more understandable
80	wow, same as before!
84	all snippets are still readable. no preference so long as the conditional statements are straight forward .
85	It is more succinct.

Field summary for ConditionSnippets [1]

Please consider the following code snippets which all represent the same semantics and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: [Ranking 1]

Answer	Count	Percentage
Snippet 1 (A1)	13	24.07%
Snippet 2 (A2)	7	12.96%
Snippet 3 (A3)	34	62.96%

Field summary for ConditionSnippets [2]

Please consider the following code snippets which all represent the same semantics and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: [Ranking 2]

Answer	Count	Percentage
Snippet 1 (A1)	13	24.07%
Snippet 2 (A2)	28	51.85%
Snippet 3 (A3)	13	24.07%

Field summary for ConditionSnippets [3]

Please consider the following code snippets which all represent the same semantics and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: [Ranking 3]

Answer	Count	Percentage
Snippet 1 (A1)	28	51.85%
Snippet 2 (A2)	19	35.19%
Snippet 3 (A3)	7	12.96%

Field summary for ConditionFree

Please explain your choice.

Answer	Count	Percentage
Answer	45	83.33%
No answer	9	16.67%

ID	Response
1	snippet 2 doesn't introduce unnecessary variables ot if statements
3	In general I think reducing the amout of indentation and the amount of parens is always good. Though the Snippets are small enought that it does not really matter.
6	Snippet 3 highlights the requirement for conjunction well, with each condition displayed clearly. Snippet 2 implicitly indicate conjunction well due to nested ifs, which comes at the cost of additional indentation. Snippet 1 personally is the messiest at first glance.
7	Snippet 3 works towards me: The relationship between the conditions is clear. I'd then continue, try to understand and then rename them for more expressiveness. For me, snippet 1 would be one manual step away from snippet 3 -> I'd associate vars and then continue with the same approach used with snippet 3. Snippet 2 simply sucks: it adds unnecessary nesting and makes reading it so hard, as the conditions are non-trivial an confusing by theirselves
8	clarity
10	in snippet 1 I get lost in long condition. snippet 2 and 3 tie in ease of following logic
12	2 is a bit too much nesting for just a single condition, 1 and 3 are reasonably readable (comparable).
13	The extraction of the conditional components in Snippet 3 makes it easier than Snippet 1 to understand the result of the condition. Snippet 2 also does this, but introduces mare nesting, which can be harder to read if the code inside is also complex and adds additional nesting.
14	I think my preferences of the 3 snippets are not really very far apart, but I least like snippet 2 due to the nesting, and probably prefer snippet 1 as that would be closest to how I would write the code.
16	3: wordy but easy to read 2: easy to read but very nested 1: not nested but difficult to read
17	If the third condition was more complex, I'd prefer snippet 2 over 1, but in this case it's simple enough to be easily readable in snippet 1. Snippet 3 is very unnatural, snippet 2 is fine but the nesting is unnecessary.
18	Tidy
19	Code is most verbose and has semantics
22	Easiest to understand
23	Most prefer snippet 1 as I like to have the conditions in one view. Snippet 3 helps break down the analysis, but I would prefer everything in one place. Least prefer snippet 2 as I don't like nested if statements.
26	Snippet 3 allows user to rename the conditions to make it clearer. Snippet 2 is clearer than 1 since it breaks up the conditions.
29	Snippet 3 most succinctly describes the conditional arguments in an intuitive manner, stating each condition before chaining them together. For similar reasons, Snippet 2 is preferred over snippet 1 despite containing nested if-else statements
30	I dislike having too many conditions in one line, so Snippet 1 is my least favourite. Snippet 3 is my favourite as it clearly marks out each of the 3 conditionals, allowing me to consider them more easily.
31	I am used to Snippet 1 where all conditions are crammed into one "if", but I like Snippet 3 for extracting booleans that I can label. For Snippet 2, nesting is unnecessary when there is no else case.
32	snippet 3 is way easier to read because the conditions for the if-statement is very clearly

	defined.
34	top choice makes it clear that we have 3 conditions, what they are, and that all have to be true. Bottom choice has too much in one line
37	Sufficient verbosity and readability tradeoff
38	Snippet 1 is harder to read. Snippets 2 and 3 are both OK, I guess. Snipped 2 looks better but #3 uses less nesting. I prefer #2 a bit. because storing the values in variables distracts a bit from program flow.
40	1 is barely readable, and 3 breaks everything down in seperate parts, making it easier to understand
41	snippet 3 is easiest to glance, while snippet 2 hides the inherent and-ing of the conditions
42	I actually would prefer snippet 4 here (inspired by what the standard code formatter for the Rust language does): if((a == 5) && (c
43	
47	I like the concise if statement of Snippet 3. The conditions are simple enough that I prefer their conjunction over nested if statements.
48	Clean
50	For Snippet 1, there are too many conditions and parentheses and the reverse engineer might get lost or read something wrongly due to operator's precedence. For Snippet 2 and 3, it breaks the condition into smaller chunks and is easier to understand.
51	Snippet 2 is more readable and easy to understand. Snippet 1 requires me to understand all the condition at once which can be quite confusing. Snippet 3 requires me to calculate the conditions before going into the if statement.
52	Snippet 3 seems easiest to follow and know the conditions use distinct variables. Snippet 1 is ok (for this example), can tell it's "all or nothing" Snippet 2 for me, nested ifs are visually hard to read (bracing for nested elses)
53	I dislike nested instructions. Snippet 1 and 3 looks cleaner although i would much prefer snippet 1 as it looks more like a proper source code.
54	Snippet 3 is much easier to read compared to the first 2. In this case, I might be accessing each of the conditionals separately. Having each of the conditionals as separate intermediate variables lets me rename them accordingly. I can do something similar in snippet 2 by commenting on each line. Snippet 1 is the worst because inlined conditionals make it difficult mark which check is doing what.
56	Fewer lines of code, especially when there are no else clauses for each of the conditions, are preferred as they help make code more readable and quicker to understand.
58	Snippets 3 & 2 are tied. They make reading the conditions necessary to enter the loop easier to interpret.
62	Snippet 1 has a single but very complicated boolean statement, so that is hard to understand. Snippet 2 is easy to read, but the longer the statement gets, the more nested the function will be. At some points, very nested function also might be difficult to understand. I like snippet 3 the most, as it consist only of few, simple boolean functions, which are eventually combined by simple operations (AND).
63	Snippet 1 seems most familiar and easiest at first glance, but the nested parentheses are hard to read. Snippet 2 and 3 are easy to read. While the nested if statements in 2 would probably become harder to read if there were even more of them, one only has to read them once when looking through the code, whereas in snippet 3 in line 5 one has to remember the contents of lines 2-4.
64	Thinking about visual flow, snippet 3 has the best presentation of the conditions, while snippet 2 at least breaks them also down on individual lines.
78	having all the conditions in one line is too much
79	Snippet 3 looks cleaner without the nested if else compared to snippet 2 & easier to evaluate compared to snippet 1
80	Snippet 3 is very easy to read and I have the opportunity to rename the three `cond` variables, hence it's ranked highest. Snippet 2 is deeply nested just for the sake of a logical and, hence I prefer Snippet 1 over it even though Snippet 1 has a hard-to-parse arithmetic expression (it was a close call between 1 and 2, both are relatively hard to read).
82	Snippet 3 is easier to read.
84	makes it easier to follow conditions
85	They are ranked in order of clarity of the conditions. Nested if-else are hard to read.

Field summary for ForLoopSnippets [1]

Please consider the following code snippets which all represent the same code and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: Snippet 4: [Ranking 1]

Answer	Count	Percentage
Snippet 1 (A1)	17	31.48%
Snippet 2 (A2)	14	25.93%
Snippet 3 (A3)	17	31.48%
Snippet 4 (A4)	6	11.11%

Field summary for ForLoopSnippets [2]

Please consider the following code snippets which all represent the same code and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: Snippet 4: [Ranking 2]

Answer	Count	Percentage
Snippet 1 (A1)	7	13.21%
Snippet 2 (A2)	19	35.85%
Snippet 3 (A3)	17	32.08%
Snippet 4 (A4)	10	18.87%

Field summary for ForLoopSnippets [3]

Please consider the following code snippets which all represent the same code and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: Snippet 4: [Ranking 3]

Answer	Count	Percentage
Snippet 1 (A1)	10	18.87%
Snippet 2 (A2)	11	20.75%
Snippet 3 (A3)	10	18.87%
Snippet 4 (A4)	22	41.51%

Field summary for ForLoopSnippets [4]

Please consider the following code snippets which all represent the same code and rank them from your favourite to least favourite. Snippet 1: Snippet 2: Snippet 3: Snippet 4: [Ranking 4]

Answer	Count	Percentage
Snippet 1 (A1)	19	35.85%
Snippet 2 (A2)	10	18.87%
Snippet 3 (A3)	9	16.98%
Snippet 4 (A4)	15	28.30%

Field summary for ForLoopFree

Please explain your choice.

Answer	Count	Percentage
Answer	44	81.48%
No answer	10	18.52%

ID	Response
1	the update on the while loops variable seems the most understable
3	It somewhat depends on what the actual code is. If there is a variable in the code that caculates x and y I might be persuaded to favor Snippet 3.
6	Assuming the incrementation amount is not affected by Block #1, it would be best preferred for the amount to be in the for loop.
7	Snippet 2 feels most natural for me. Snippet 3 is a nice and short for-loop, though I don't like updating x and y in the inner loop scope. Snippet 1 is LESS expressive than 3, but replaces the x-y update of snippet 3. Yet snippet 3 wins over 2 because of overall clarity. Snippet 4 mixes 1 and 3 and is "all over the place". I have to watch out for non-trivial variable updates in both the inner loop and the official iteration update. No bueno.
8	clarity
10	I prefer easy loop headers (with complexity in body)
12	I think the update is not complicated enough to warrant splitting out of the loop head. But if it is split out, i prefer 2 the most as everything is in one place. 3 and 4 are close, maybe tied.
13	I think a for-loop is easier to read in this case as a while-loop. And to extract a complex operation from the head of the loop increases the readability of the loop.
14	1 and 2 are quite clear, and I personally feel for-loops are easier to read. 4 and 3 tends to be a bit confusing as the for-loop operation occurs within the for-loop itself.
16	1: it's a bit difficult to read, but easy to see, that i is calculated without other dependencies 2,3,4: more easy to read but the additional variable(s) in the loop and different places of calculation complicates up reading it.
17	Having the loop increment statement reference variables that are set in the loop is hard to follow (snippet 4 and 3), having a while loop instead of a for loop can also be harder to read (You have to watch out for continues to make sure the loop counter changes in each iteration).
18	Neat
19	Snippet 3 was easily understood, while snippet 4 required determining the logic within the for loop. Snippet one was essentially code-golfing (not essential), and while loop is harder to understand without extra effort into anlysing it.
22	Closest to how I would implement in C
23	Prefer for loops the most as I could explicitly see when the loop terminates. Prefer the for loop in snippet 1 the most as I could easily see how the loop variable i updates in one statement.
26	For statement is more concise that while statement. Separating the conditions into x and y variable allows user to rename them.
29	Snippet 1 most clearly states the conditions of the for loop in a single sentence without additional confusing variables. Snippet 2 is deemed the worst since the changes in c can easily be lost in the block of code in between
30	Snippet 1 least favourite because it is all in one line, hardest to parse. Snippet 2 and 4 is next because I prefer to see more than 1 variable to manage the loop counter, but 4 is a bit confusing still. Snippet 3 is the cleanest to me.
31	Snippet 4 seems to have the best balance between simplifying the update statement and introducing new variables. I prefer for loops to while loops, as it's easier to find the update statement(s).
32	putting the loop exit condition logic together makes it easier to read the code. It gets harder/more annoying when they are separated like in Snippet 4 (x - (i >> 32)). Then I have to look at what is 'x' at the bottom of Block #1. Then it turns out that x is related to i as well... Might as well put them together.
34	I hate for loops with anything else but i++ or i--

37	How easy it is to understand the logic
38	Having complex operations inside the for loop definition is a bad idea IMHO. What is more, I think while loops have better readability most of the time. By substituting most of the operations, #3 is "the best of the rest". P.S.: Why the `>> 5 >> 2`? Isn't that just the same as `>> 7`?
40	i prefer the least amount of logic possible inside the loop definition
41	snippet 1 is too complex in the for header. I generally prefer for-loops over while, but with a complex update such as this I prefer the while loop. snippet 3 is okay but I feel like the update of a for-loop should be self-contained, i.e. not partially in the for-header and partially in the body. Snippet 4 has the same problem as snippet 3 but is also more complex in the for-header
42	Snippets 3 and 4 split the loop variable handling between the top and the bottom of the loop, which is bad. Snippets 1 and 2 have the same readability to me, here I can only say on a case-by-case base which one I prefer.
47	I prefer Snippet 1 and 2, which keep the statements involving the loop variable 'together'.
48	Simpler to understand
50	For Snippet 2, the While Condition is very clear and one needs to only calculate c and check if it is greater than 10.
51	Snippet 3's for loop has an easy to understand (x-y) update statement, although the calculations for x and y are rather confusing Snippet 2 is a simple one-liner that is also easy to understand Snippet 4 has a rather intimidating (x - (i >> 32)) update statement, which is made worse when x also has a rather confusing calculation. Snippet 1 looks the most unpleasant because the update statement is long and confusing.
52	Personally prefer if the loop looks as simple as possible? For Snippet 2, at least the calculation has its own line Prefer 1 over 3 and 4, to avoid splitting across multiple lines and adding new variables
53	Snippet 1 is simply too complicated. It is easier to understand the loop condition from snippet 3 and 2.
54	Snippet 3 is much easier to process by creating intermediate variables x and y and operating on them within the loop brackets. Snippet 2 is second, with a similar reason as above. Snippet 4 is similar to 3, but not as good because it only creates 1 intermediate variable x - think it is better to commit to either having the operation within the loop syntax or not. Snippet 1 is difficult to analyze because it creates a lot of clutter on the for loop syntax.
56	Snippet 1 allows quicker understanding of what the number of rounds in a for loop is dependent on. Snippet 3 is better than snippet 4 as the values for x and y are placed together, allowing for quicker deriving of how variable i is calculated.
57	Snippet 4 appears to be the most intuitive solution as one would expect a for-loop-statement to be (even though the update statement is complex). Considering that the update statement is complex, Snippet 3 is the next favorable solution: Rather simple update statement, values are determined in the for-loop. Even though Snippet 2 is no for-loop, the while-loop looks like it was written by human beings. Snippet 4 appears to be the least favorable solution as the Update-Statement is a mixture of a value determined in the loop and the former iteration value. This is a mixture of Snippet 1 and Snippet 3.
58	Snippet 1 clearly shows the for-loop's continuation and increment conditions. However, if they become too complicated, breaking them down into that shown in Snippet 3 can be handy. However, in Snippet 4, it is neither within the for-loop's brackets nor extracted fully at the end of the loop. This feels half-baked and somewhat confusing to follow. The use of while-loops in Snippet 2 requires the initialization to be done outside and before of it and is least favourable for me.
62	Shifting, especially inside the condition of the FOR-loop, is difficult to understand. So, it might be unclear, over which exact elements are iterated. Therefore, snippet 3 and 2 are the best readable snippets. I prefer snippet 3 over snippet 2 because the iteration step is a bit more clear. Snippet 1 is the in my opinion the worst as the iteration step consists of multiple complicated statements, including shifts, and is therefore hard to interpret.
63	i think the for loop is easier to read in this case. I find splitting up the i update in snippet 4 more confusing than helpful, since now one has to look in 2 places. Either snippet 3 or snippet 1 are fine. While in snippet 1 the functionality of the for loop is more clearly stated in one place, snippet 3 is easier to read. Here the structure, length and content of Block #1 would most likely also influence the readability, i.e. snippet 3 would be less favorable than snippet 1 if block #1 was long and updates to x and y somehow hidden in the block.

64	if I see "for" my first thought is some action repeated in a sequence for times. While on the other hand implies a repetition for as long as a condition is met. That makes snippet 2 feel more natural here. Snippet 1 at least keeps the condition out of the body, 3+4 are a mix where need to read and think about too many lines to understand what and why this condition is updated.
78	having too much calculations at the for loop makes the code harder to follow
79	It's easier to understand if the condition is modified outside of the loop rather than putting it all in one line
80	for loops can transport a lot more meaning by basically deciding that a certain variable is a counter. Hence I prefer snippets with for loops over the one with the while loop. I ranked the for-loop snippets by how well the decompiler as able to distinguish in-loop calculations from progression-step calculations: Snippet 4 looks the cleanest there.
84	no exact preference between snippet 1 and 3 as they are the same and easy to distinguish the conditions and the instructions after the loop. snippet 4 makes it easy to make mistakes by splitting the instructions between the code block and the for end-loop
85	Conditions of the loop are not diluted within the loop.

Field summary for feedback

Thank you very much for getting this far and also for participating in any previous surveys! This is very likely to be our last decompiler survey for now. If you would like to leave us any feedback about the survey, please use the lines below to help us improve ourselves.

Answer	Count	Percentage
Answer	11	20.37%
No answer	43	79.63%

ID	Response
1	some preferences don't necessarily only relate to the mainly investigated issue of the page
19	-
31	It would be nice to have some kind of auto-save to our survey token, as both times I accidentally exited before finishing or saving and had to redo it. Thanks for the effort and good ideas!
32	Thanks for the opportunity.
34	I feel like using the /* BLOCK x */ stuff skews perception of the snippets heavily
38	The usability was very good IMHO
42	I think I already mentioned that looking at what the standard Rust code formatter does for way too complex code constructs could also be useful for decompiler output. Sometimes smart intendation of code helps more in the way of readability than rewriting the code with a different control flow style.
62	Most of the code snippets where very similar. I would suggest to show more different examples. Furthermore, it was unclear what the intention of the code supposed to be. If more context was shown, choosing which option is preferable would be probably easier.
64	good luck on the paper submission! :)
80	very interesting survey, made me think about my preferences and why I have them!
84	great piece of work! look forward to using it to analyze codes