! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

note @365 222 views

[Exams] Past Exams 2017 Q&A

Discuss all questions pertaining to exams which took place in 2017 here.

You can find the past exams here: https://cs61c.org/resources/exams

When posting questions, you **MUST** reference the semester, exam, **AND** question so we can help you.
Please put this at the beginning of your post in this format: **[{Semester}-{Exam}]:Q{Question Number}**
For example: **[SP-MT1]:Q1**, or **[SU-MT2]:Q3**

{Semester} is one of these: SP, SU, FA
{Exam} is of of these: Q, MT, MT1, MT2, F

If you follow this format, it will make it very easy to search for similar questions!

midterm_exam1 midterm_exam2 final_exam

~ An instructor (Jie Qiu) thinks this is a good note ~

Updated 1 month ago by Stephan Kaminsky

followup discussions *for lingering questions and comments* 1 endorsed followup comment

Resolved Unresolved

Anonymous Poet 4 months ago
[FA-MT1]:Q2

I had a quesiton for Midterm 1, Fall 2017 Question 2.

The Answer key does not show any explanation I was hoping to clarify where all the variables are located.

&dictionary - STACK
&num_words - STATIC
dictionary - HEAP
&dict_size - STACK
&word1 - STATIC
&dict - STACK
dicitonary[1] - HEAP
dictionary - HEAP
word1 - STATIC
&(word2[1]) - CODE
*dictionary - HEAP

Can someone clarify if my list is right? When answering parts 1-7, I noticed that I got #3 and #6 incorrect so I am not sure what to say for the following variables:

&word1 - ???
&dict - ???
&(word2[1]) - ???

```
#define MAX_WORD_LEN 100
int num_words = 0;
void bar(char **dict) {
      char word2[] = "BEARS!";
      dict[num_words] = calloc(MAX_WORD_LEN, sizeof(char));
      strcpy(dict[num_words], word2);
      num_words += 1;
}
int main(int argc, char const *argv[]) {
      const int dict_size = 1000;
      char **dictionary = malloc(sizeof(char *) * dict_size);
      char *word1 = "GO";
      bar(dictionary);
      bar(dictionary);
      return 0;
}
```

helpful! | 0

Anonymous Poet 4 months ago nvm resolved

helpful! | 0

[REDACTED] 4 months ago I have the same question. How did you get 3?

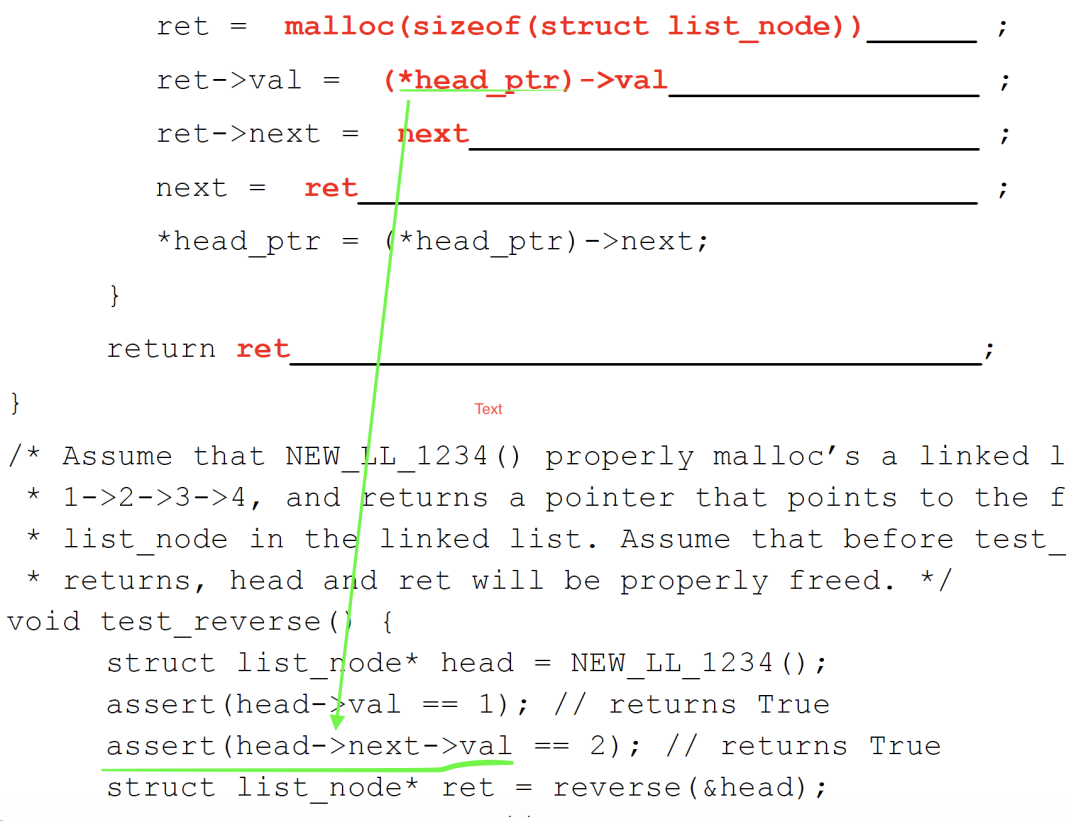
helpful! | 0

Resolved Unresolved

Anonymous Poet 4 months ago
[FA-MT1]:Q3

Why did the head pointer need to be de-referenced in the function call of reverse() but not in test_reverse() to obtain the value of the node? (the line segements corresponding to the green arrow and green underlined)

Why couldn't the answer to the line for **ret->val** just be **head_ptre->value;**



helpful! | 0

Anonymous Atom 4 months ago I believe this is because in test_reverse, head is a single pointer struct list_node* head, whereas in reverse, head_ptr is a double pointer struct list_node ** head_ptr, and one can see that it is passed in as &head, or a pointer to head.

helpful! | 0

Resolved Unresolved

Anonymous Poet 4 months ago

[FA-F]:Q9

I had a question concerning Question 9 from Fall 2017 Final exam. I do not understand what y represents in the answer for part 2. I understand since there are 7 mantissa bits, then we increment by 2^{-7} between values in the same exponent representation. Why is the answer 2^{y-7} and how was y incorporated in the answer key when it is possible to "jump" into a higher exponent, resulting in large gaps between numbers being represented?

- The smallest exp follows the same denormalized formula as traditional floating point
- **Numbers are rounded to the closest representable value. Any numbers that have 2 equidistant representations are rounded down towards zero.**

All of the parts of this question refer to this floating-point scheme.

1. What is the smallest nonzero positive value that can be represented? Write your answer as a numerical expression in the answer packet.

2^{-13}

2. Consider some **positive normalized** floating-point number p where p is described as p = 2^y * 1.significand. What is the distance (i.e. the difference) between p and the next-largest number after p that can be represented? Write your answer as a numerical expression.

2^{y-7}

helpful! | 0

Jie Qiu 4 months ago The significand indeed increases by 2^{-7} but you also have to take into account the fact 1.significand is multiplied by 2^y. $2^y \times 2^{-7} = 2^{y-7}$.
also what do you mean by "'jump' into a higher exponent"?
helpful! | 0

Resolved Unresolved

Anonymous Mouse 4 months ago
[FA-MT1]:Q2

I was a bit confused as to wear the different variables were located.

1. &dictionary -- stack because the pointer is created in the stack? Just dictionary would be heap?

2. &word would be stack since pointer is in the stack but just word would be in static?

Also, 3. When we do something like char ** dictionary = malloc(...) what does this look like in memory? Is it a pointer to a pointer to a block of memory? How are we allowed to index into dictionary in the void bar() function?

helpful! | 0

Jie Qiu 4 months ago 1. Yes and yes. dictionary is a local variable.
2. Yes
3. Yes it's a pointer to a pointer, which is essentially an array of pointers. Thus dict[index] returns a char *.
helpful! | 1

Anonymous Mouse 4 months ago Great thanks! I was just a little confused because comparing to HW 2:

dictionary from the midterm q is in the heap but here, *chekov* is in the stack?

```
#define SPOCK 1701
int KIRK = 1701
int sulu(int scotty) {
    return scotty * scotty;
}
int main(int argc, char *argv[])  {
    int *chekov = malloc(sizeof(int) * 1701);
    if (chekov) free(chekov);
    sulu(SPOCK); //  ← snapshot just before it returns
    return 0;
}
```

helpful! | 0

Albert Magyar 4 months ago Here, `chekov` is a local variable that has pointer type `int *`. Therefore, `chekov` is stored on the stack.

Since `chekov` is a pointer, it has a value that is the address it points to. This address points to an area on the heap, since it is set to the return value of `malloc`.

Pointers contain addresses, but they themselves must be stored somewhere. The variable `chekov` lives on the stack and points to the heap.

helpful! | 0

Anonymous Mouse 4 months ago That makes sense but using the same logic, shouldn't *dictionary* from the midterm problem also be stored on the stack since it is a pointer? (the instructor above/problem sol said heap)

(*chekov) points to memory so it is on the heap. In the same way shouldn't (**dictionary) point to the heap (not just dictionary)??

helpful! | 0

Jie Qiu 4 months ago The variable dictionary is on stack. The value of dictionary tho, is a heap address. **dictionary and *checkov are located on the heap, and have very specific values that are not memory addresses. For example, *checkov has the value of some integer.

helpful! | 0

Anonymous Mouse 4 months ago Ok but in your earlier answer, I asked if just dictionary would be on the heap and you said yes. But now you are saying dictionary is on the stack? I'm very confused : (

helpful! | 0

Daniel Li 4 months ago Dictionary is a pointer that exists on the stack. Dictionary points to values that (i.e *dictionary) exists on the heap. This is what is meant when it says the value of dictionary is a heap address

helpful! | 0

Resolved Unresolved

Anonymous Mouse 4 months ago

[FA-MT1]:Q3

Why did we have to pass in head_ptr as a ptr to a ptr?

helpful! | 0

Anonymous Mouse 4 months ago Also, how is there a memory leak?

helpful! | 0

Anonymous Poet 4 months ago did you draw everything out

helpful! | 0

Jie Qiu 4 months ago To your first question, we want to modify a pointer so we have to pass in a pointer to pointer. This

helpful! | 0

Anonymous Mouse 4 months ago Oh okay that makes sense. What would have gone wrong in the code in we just passed in a regular pointer?

helpful! | 0

Albert Magyar 4 months ago Since the reverse function returns a new copy (and does not modify what the pointer-to-pointer points to), it would have been fine in theory to write a reverse-as-copy function that accepted an argument of type `struct list_node *`.

In this case, the decision to make the argument a pointer-to-a-pointer is an arbitrary decision. The reason it's possible to answer the question definitively is that it's clear from the body of `reverse` and the test code that the function takes a struct list_node **.

helpful! | 2

Anonymous Mouse 4 months ago Great thank you! Just last question, where is the memory leak??

helpful! | 0

Albert Magyar 4 months ago You allocate a new copy in test_reverse using reverse. Since this copy doesn't get returned from test_reverse, it becomes inaccessible when test_reverse returns; thus the memory is "leaked".

If a function performs some action (including calling another function) that allocates memory, it is a memory leak unless it does one of the following:

1. Frees the buffer
2. Returns a pointer to the buffer
3. Makes the buffer visible by storing a pointer to it in some global state
4. The function is main

helpful! | 2

Albert Magyar 4 months ago Another critical point: the past year's solution is weird. Even though it doesn't need to do this to return a reversed copy, it modifies the pointer that `head_ptr` points to and sets it to `NULL` in all cases. This has the effect that `head` in `test_reverse` ends up being set to `NULL` after reverse is called, which means that the original linked list is also leaked.

Generally, writing questions that have you complete "broken" code can be a tricky thing to balance; I would say that that problem is a bit too weird, since it is actually possible to fill in the blanks in the code such that the code has more sensible behavior. The problem makes it hard to tease out what the "spec" of the code is.

helpful! | 2

Anonymous Mouse 4 months ago That makes sense, thanks!

Regarding your first message, the problems says "Assume that before test_reverse* returns, head and ret will be properly freed." Thus the new copy in test_reverse using reverse isn't leaked right? Or is there something else I am missing.

helpful! | 0

[REDACTED] 4 months ago I think the problem says "before test_reverse returns, head and ret will be properly freed" which makes the memory leak Albert Magyar says incorrect. Am I wrong?

helpful! | 0

Albert Magyar 4 months ago The `reverse` function modifies `head` (since it gets a pointer to it and changes what that points at). Therefore, the original buffer that `head` pointed at, which was allocated by the function to create the 4-element list, is now inaccessible. Its address has been lost, so there is **no way** for it to be properly freed.

helpful! | 0

Albert Magyar 4 months ago Again, the way this question asks you to implicitly juggle the code's intended behavior and actual behavior with few cues makes it a bit on the "weird" side, and the solutions contain very little explanation to help your intuition. The fact that the memory leak is an apparent contradiction with the statement that "before test_reverse returns, head and ret will be properly freed" is an example of how the question is perhaps not as unambiguous as it should be.

helpful! | 0

[REDACTED] 4 months ago It's very clear now. Thanks!

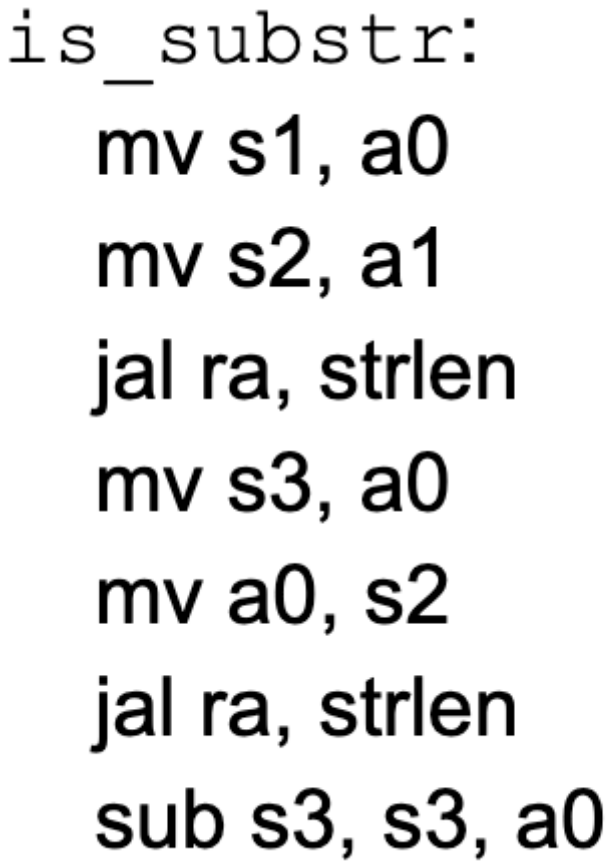
helpful! | 0

(•) Resolved ( ) Unresolved

Anonymous Mouse 4 months ago

[FA-MT1]:Q5

I had a question about calling the strlen function using jal. Where is the result of jal stored? I assumed it was a0 but is this a standard convention? I remember hearing something but if someone could give me some clarification that would be great!



helpful! | 0

Daniel Li 4 months ago Yes we assume that return values are always stored in a0 by convention, so here, the return value of strlen will be in a0.

helpful! | 0

Resolved Unresolved

[REDACTED] 4 months ago

[FA-MT1]:Q2

2.3 &word1 evalutes to static address. What value does &dict evaluate to that it's smaller than &word1?

helpful! | 0

Daniel Li 4 months ago For 2.3, we are looking at &word1 and the the pointer for word1 is allocated on the stack frame of main so it is on the stack. &dict is the address of dict. Since dict is an argument for the function bar, which is called after word1 is initialized, the address of it be on the stack but below word1 since the stack grows downwards

helpful! | 0

[REDACTED] 4 months ago Thanks. Get it.

helpful! | 0

Resolved Unresolved

Anonymous Beaker 4 months ago

[FA-F]:Q1

I don't understand how this particular part of the answer works here: (m & 0xEEEEEEEE) == 0

helpful! | 0

Sruthi Veeragandham 4 months ago Any power of 16 will have a 1 in the last "slot" of the hex digit -- 16 (16^1) is 0b0001_0000 or 0x10, 256 (16^2) is 0b0001_0000_0000 or 0x100, and 4096 (16^3) is 0x1000. This means that if you AND a power of 16 with 0xEEEE_EEEE, you should get 0 -- (since 0xE is 0b1110). You'd also get a 0 if you ANDed 0xEEEE_EEEE with 0, which is why there's also a check that m is not 0.

There are some non-powers of 16 m for which m & 0xEEEE_EEEE == 0, for example 0x0000_0011 (272). However, the second check in the if statement (m & (m - 1) != 0) eliminates those cases. For any power of 2 represented in binary as 1000...0, that power of 2 minus 1 will be 0111...1, so for any power of 2, m & (m - 1) will be 0, and a power of 2 will have exactly one 1 in its binary representation.

~ An instructor (Jie Qiu) thinks this is a good comment ~

helpful! | 2

Resolved Unresolved

Anonymous Calc 1 month ago

[FA-F]:Q4.5

Can someone explain how entering 21 for user_in_2 prints out what's stored in Skraa? It seems like what's stored at the address of s1/Boom has to change somewhere in lines 14-20 but I'm not sure how that happens exactly.

helpful! | 0

Anonymous Comp 1 month ago I'm not totally sure, but this is how I interpreted it:
In lines 14-16, you store the address of the instruction at line 21 to s0.
On line 17, you load the machine instruction of line 21 (addi a1, s1, 0) into t1.
On line 18, you shift the user input left by 20 so that it aligns with the immediates section of a type I machine instruction (bits 31-20, if you look at the RISC-V sheet).
On line 19, you add the user input and the previously loaded machine instruction to get a new machine instruction that has a new imm[11:0] value.
In line 20, you store this instruction into memory, thereby replacing (addi a1, s1, 0) on line 21.

Hence instead of addi-ing the address of Boom in a1 on line 21, you have a1 = (address of Boom) + 21, which is the address of Skraa apparently.
However, I don't get why 21 is the correct offset? Why don't we multiply 21*4 since every character is 4 bits?

helpful! | 0

Anonymous Comp 1 month ago ^do la and ecall use pointer-sized addresses maybe?

helpful! | 0

Resolved Unresolved

Anonymous Comp 1 month ago

[FA-F]:Q4.2

How do we get the imm[4:0] field to be 0b10000? I thought the offset would be (8 lines)*4 = 32. At first, I thought that maybe the zeroeth bit gets cut off. But from past assignments, it seems like the zeroeth bit is only cut off when the construction format explicitly does so (e.g imm[4:1|11] for SB types).

helpful! | 0

Anonymous Comp 1 month ago The solution would actually make sense if we were using SB type instruction formats, although the table labels seem to suggest S-type instruction formats...

helpful! | 0

Charles Hong 1 month ago

Sorry about the incorrect solution. First of all, as you said, `beq` is an SB-type instruction, so the table should have labels imm[12|10:5] and imm[4:1|11]. Even if this were corrected, for the correct offset of 32 bytes we would need those fields to be 0b0000001 and 0b00000 respectively. The implicit 0 bit appended to the end for SB-types takes the place of bit 0, which is not included in the machine code. Good catch!

helpful! | 0