Fall2021-CS33 midterm

MATTHEW FIORELLA

TOTAL POINTS

54.5 / 62

QUESTION 1

Question 1 14 pts

1.1 2 / 2

✓ **- 0 pts** correct

- 2 pts incorrect or blank, correct answer b and d. No partial credit available.

1.2 0 / 2

- 0 pts Correct

✓ **- 2 pts** incorrect or blank answer , correct answer is c,d

1.3 2 / 2

✓ **- 0 pts** Correct

- 2 pts incorrect , answer should be blank

1.4 0 / 2

- 0 pts Correct

✓ **- 2 pts** incorrect or blank, correct answer should be e

1.5 2 / 2

✓ **- 0 pts** Correct

- 2 pts incorrect

1.6 2 / 2

✓ **- 0 pts** Correct

- 2 pts Incorrect

1.7 2 / 2

✓ **- 0 pts** Correct

- 2 pts Incorrect

QUESTION 2

Question 2 8 pts

2.1 2 / 2

✓ **- 0 pts** Correct

- 2 pts incorrect, answer should be 32

2.2 2 / 2

✓ **- 0 pts** Correct

- 2 pts incorrect or blank, the correct answer should be 16

2.3 2 / 2

✓ **- 0 pts** Most

- 2 pts Least

- 2 pts Other Wrong Answer

2.4 0 / 2

- 0 pts Correct

- 2 pts Incorrect

✓ **- 2 pts** float*

- 2 pts char

- 2 pts Blank

- 1 pts Vague: both

QUESTION 3

Question 3 5 pts

3.1 Zero 1 / 1

✓ **+ 1 pts** Correct

+ 0 pts Incorrect

3.2 One 1 / 1

✓ **+ 1 pts** Correct

+ 0 pts Incorrect

3.3 Smallest denormalized number other than 0 1 / 1

✓ **+ 1 pts** Correct

+ **0 pts** Incorrect

3.4 Smallest possible normalized number **1 / 1**

✓ + **1 pts** Correct

+ **0 pts** Incorrect

3.5 NaN **1 / 1**

✓ + **1 pts** Correct

+ **0 pts** Incorrect

QUESTION 4

Question 4.1 6 pts

4.1 a **1 / 1**

✓ - **0 pts** Correct

- **1 pts** Wrong

4.2 b **1 / 1**

✓ - **0 pts** 1

- **1 pts** not 1

4.3 c **1 / 1**

✓ - **0 pts** 8

- **1 pts** not 8

4.4 d **1 / 1**

✓ - **0 pts** Correct

- **1 pts** not 2

4.5 e **1 / 1**

✓ - **0 pts** Correct

- **1 pts** not 40

4.6 f **1 / 1**

✓ - **0 pts** Correct

- **1 pts** not 2

QUESTION 5

5 Question 4.2 **2 / 2**

✓ - **0 pts** Correct

- **1 pts** array dimension $$[2][5]$$

- **1 pts** should be array of two pointers

- **0.5 pts** missing int

- **1 pts** location of pointer symbol

- **2 pts** wrong/blank

- **1.5 pts** not just words, need C declaration

QUESTION 6

6 Question 5a **6 / 6**

✓ - **0 pts** Correct

- **2 pts** Wrong sub/add amount or no sub/add

- **1 pts** extra instructions

- **1 pts** Missing push/pop

- **4 pts** No push/pops

- **1 pts** Missing add/sub

- **1 pts** Wrong order Prologue

- **1 pts** Wrong order epilogue

- **1 pts** Missing second push/pop

- **6 pts** Blank / Not correct

- **1 pts** add/sub swapped but correct amounts

- **1 pts** pushing/popping wrong thing

- **1 pts** pushing/popping wrong thing x2

- **1 pts** misusing leaq instead of add/subq

QUESTION 7

7 Question 5b **3 / 4**

- **0 pts** Correct

- **2 pts** No subtract/add

✓ - **1 pts** **Not enough subtracted/added from stack**

- **1 pts** extra operations

- **1 pts** Wrong order

- **1 pts** missing push

- **1 pts** missing pop

- **1 pts** Impossible subtraction/addition (not multiple of 8)

- **4 pts** Wrong / Blank

- **1 pts** Missing corresponding sub/add or amount mismatch

- **1 pts** misusing leaq instead of add/sub

QUESTION 8

Question 6 7 pts

8.1 Blank 0 0.5 / 1

+ **1 pts** Correct

✓ + **0.5 pts** **Partially correct**

+ **0 pts** Incorrect

8.2 Blank 1 1 / 1

✓ + **1 pts** **Correct**

+ **0.5 pts** Partially correct

+ **0 pts** Incorrect

8.3 Blank 2 1 / 1

✓ + **1 pts** **Correct**

+ **0.5 pts** Partially correct

+ **0 pts** Incorrect

8.4 Blank 3 1 / 1

✓ + **1 pts** **Correct**

+ **0.5 pts** Partially correct

+ **0 pts** Incorrect

8.5 Blank 4 1 / 1

✓ - **0 pts** **Correct**

- **1 pts** Incorrect

- **0.5 pts** Partially incorrect

8.6 Blank 5 1 / 1

✓ - **0 pts** **Correct**

- **1 pts** incorrect

- **0.5 pts** Partially incorrect

8.7 Blank 6 1 / 1

✓ - **0 pts** **Correct**

- **1 pts** Incorrect

QUESTION 9

9 Question 7 4 / 4

+ **1 pts** Infinity&Nan

+ **0.5 pts** Only Zero or Only Infinity or Only NaN

+ **0 pts** Incorrect Case

✓ + **1 pts** **Literally true but without being insightful**

✓ + **3 pts** **Functionality Correct**

+ **0.5 pts** Many True Statements

+ **0 pts** Incorrect Function

+ **0 pts** Funniness Bonus

+ **1 pts** Normalizes Floating Points

+ **2.5 pts** Right but what's an unsigned float?

QUESTION 10

10 Question 8 6 / 6

✓ + **1 pts** **Binary Search Tree**

+ **0 pts** Wrong Datastructure

✓ + **5 pts** **11 or b (or 12)**

+ **0.5 pts** Good Try!

+ **0 pts** Good Try...

+ **0 pts** Blank

+ **0 pts** Funny answer

+ **4 pts** Sooo close!

+ **5 pts** close enough!

+ **3 pts** There's definitely some good stuff here!

CS33: Intro Computer Organization
Fall 2021 Midterm

Name: Matthew Fiorella

UID: 605593729

(Please write as legibly as possible!)

This is an open book, open notes exam, but you cannot share books/notes. Please follow the university guidelines in reporting academic misconduct.

Note: If the architecture of the machine is not specified, assume that the question is being asked in the context of a 64-bit little endian x86 machine, running Linux.

Please wait until everyone has their exam to begin. We will let you know when to start.

Good luck!

	Points Possible
1. Multiple Choice	14
2. Baba is Struct	8
3. Notable Floats	5
4. Array Interpretation	8
5. Imaginary Stack Allocator	10
6. Tricky Switch	7
7. Floating Point Mystery	*4*
8. Two Birds, One Bomb	*6*
Total	50 +12 bonus points

Question 1. Multiple Choice (14 points)

For the following multiple choice questions, **select all answers that apply.** If none are correct, leave the question blank. Put your answers in the table on the next page. (2pts each, no partial)

1. What is a difference between unsigned and signed integer representations?
   a. Unsigned integers can store a wider range for the same number of bits.
   [✓ b.] Right-shifting an unsigned integer uses "logical" shift, while right-shifting a signed integer uses "arithmetic" shift.
   c. In a C expression that operates on two different datatypes, an unsigned datatype will take precedence over a floating-point datatype, but a signed datatype will not.
   [✓ d.] It is meaningful to ask if a signed number is greater-than or equal to zero, while the same is meaningless for unsigned numbers.

2. Suppose you use the objdump command to disassemble a function, and you see this:

```
0000000000400595 <func>:
  400595:   53              push    %rbx
```

   What does the "53" in the above line represent?
   a. The function has a length (in terms of instructions) of 53 bytes.
   b. The push instruction is 53-bytes offset from the beginning of the function.
   c. "53" contains an encoding of the register %rbx.
   [✓ d.] "53" contains an encoding of the push operation.
   e. Address 53 is the location of where a callee-saved register is pushed.

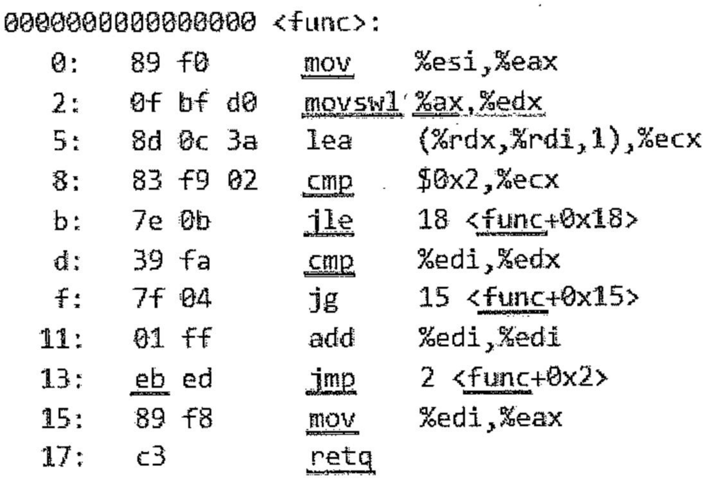
3. X86_64 contains an instruction which performs a conditional move -- **cmov** --, which moves one register to another based on the condition codes (aka. flags). This instruction can sometimes be used to perform if-statement control flow. When is performing if statements using conditional moves a better option than using ordinary branch instructions?
   ~~a.~~ When the body of the if statement contains function calls.
   ~~b.~~ When the body of the if statement contains side effects.
   ~~c.~~ When the body of the if statement contains many instructions.

4. X86_64 contains an instruction which performs an indirect jump -- **jmp*** --, which jumps to a location specified in a memory table. This instruction can sometime be used to perform switch-case-statement control flow. When is performing switch statements using indirect jumps a better option than using ordinary branch instructions?
   a. When there are many fall-through cases in the switch statement.
   [✓ b.] When there are few instructions in the case statements.
   c. When there are many instructions in the case statements.
   [✓ d.] When the first case value is zero.
   [✓ e.] When the range of case values is contiguous.

5. Suppose that the next C language standard contains an 11-bit unsigned integer datatype. What would be a valid reason (or reasons) to reject this proposal?
   a. There are already larger and smaller datatypes (e.g. 8-bit and 16-bit), so it's not useful.
   [✓ b.] Modern ISA use byte-addressable memory, so accessing arrays of contiguous 11-bit integers would require extra instructions to extract the number.
   c. For a two's complement number to be well-defined, its size must be a multiple of 2.
   d. Having an 11-bit number would make it impossible to satisfy the datatype casting rules in the C standard.

6. For what C datatypes is the concept of "endianness" irrelevant?
   [✓ a.] char
   [✓ b.] unsigned char
   [✓ c.] string (i.e. array of char)
   d. int
   e. float

7. What kind of control flow is contained in this assembly function? (see figure below)
   [✓ a.] Loop (e.g. while/for)
   [✓ b.] Conditional Branch (if/else)
   c. Indirect Branch (switch/case)



```
0000000000000000 <func>:
   0:   89 f0          mov     %esi,%eax
   2:   0f bf d0       movswl  %ax,%edx
   5:   8d 0c 3a       lea     (%rdx,%rdi,1),%ecx
   8:   83 f9 02       cmp     $0x2,%ecx
   b:   7e 0b          jle     18 <func+0x18>
   d:   39 fa          cmp     %edi,%edx
   f:   7f 04          jg      15 <func+0x15>
  11:   01 ff          add     %edi,%edi
  13:   eb ed          jmp     2 <func+0x2>
  15:   89 f8          mov     %edi,%eax
  17:   c3             retq
```

Answer Table (list any correct answers)

1	bd.
2	d.
3	
4	bde.
5	b.
6	abc.
7	ab.

Question 2. Baba is Struct (8 Pts)

Consider the following structure, union, and array definitions:

```
typedef struct {
  int baba;          4
  short flag[5];    10
  float* keke;       8
  char key;          1
} noun;

typedef union {
  int hot;
  short shut[5];
  float* stop;
  char open;
} property;

noun you[5];
property me[5];
```

```
// Note: Typedef here just means
that were are defining a struct
type that we can use later in the
array definition.
```

4 10 2 8 1 7 32

baba flag pad keke key pad

```
// Create an "arrays of structs"
and an "array of unions"
```

1. If &you == 0 (i.e. if the address of you[0] is zero), at what address is you[1]?

   32

2. If &me == 0 (i.e. if the address of me[0] is zero), at what address is me[1]?

   16

3. If we access "property.shut[1]", that will also access half of the bits in "property.hot". Will that be accessing the least significant bits of "hot" or the most significant?

   most significant

4. Consider all the primitive data types within the arrays "you" and "me". Which of these are guaranteed to have their addresses aligned to a multiple of their size?

   float*

Question 3. Notable Floats (5 Pts)

The following table shows a number of "interesting" values for floating-point numbers, along with their encoding into sign, exp, and frac fields.

Pattern	sign	exp	frac
A	0	00..00	00..00
B	0	00..00	00..01
C	0	00..01	11..11
D	0	00..01	00..00
E	0	01..11	00..00
F	0	11..10	11..10
G	0	11..11	00..00
H	0	11..11	11..11

Match the following definitions to the interesting numbers above (write "A", "B", etc. in the box)

0. Zero	A.
1. One	E
2. Largest Possible Demormalized Number	~~A~~ B.
3. Smallest Possible Normalized Number	D.
4. Not a Number (NaN)	H.

Note: These patterns don't specify the number of bits, but that won't matter for answering the question. Assume the same strategy for representation of denormalized numbers, NaNs and infinities as other IEEE 754 standard numbers.

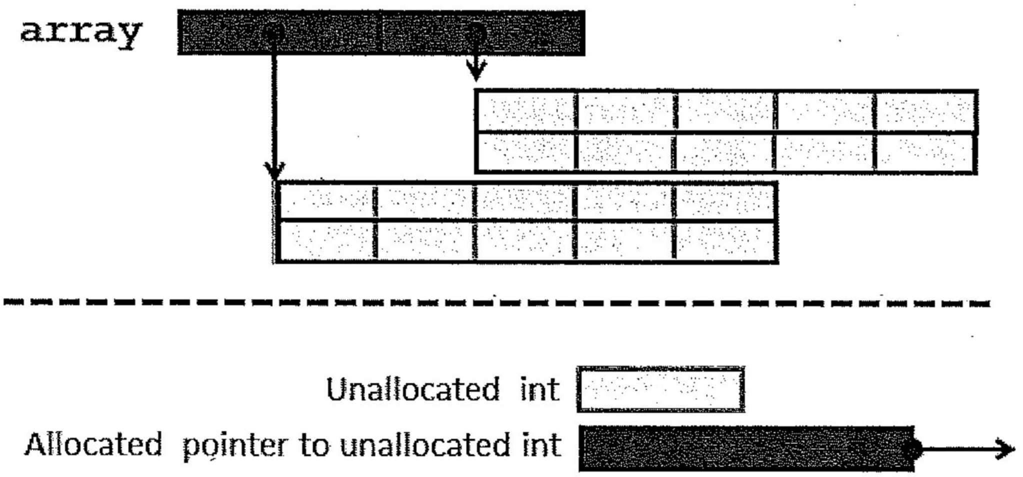
Question 4. Array Interpretation (8 pts)

1. For the following datatype definitions, answer the following questions: (6pts)

- **sizeof(array):** What is the size of "array" in terms of number of bytes? For this problem, we are only talking about the memory allocated for the variable "array", and not any other supporting data structures.
- **How many dereferences?:** List the number of memory dereferences that it would take to access an integer for that datatype. In other words, how many times do you have to access memory *total* (how many loads), to eventually access a single integer. Include the load of the integer itself.

	Array Declaration	sizeof(array)	How many dereferences?
	int* array	8	1
1	int array[3][2]	24	1
2	int (*array)[5]	8	2
3	int *array[5]	40	2

2. What is the array declaration for the following array, represented visually below? (2pts)



4	Datatype for array:	int (*array[2])[2][5]

Question 5. Imaginary Stack Allocator (10pts)

Part 1: Suppose we have two functions, FuncP and FuncQ. FuncP calls FuncQ, and the stack frames of both functions are depicted below.

All functions require a "prologue" and "epilogue" to manage the stack. The prologue allocates stack space, and usually appears at the beginning of the function. The epilogue deallocates stack space, and usually appears at the end of the function.

Based on the stack frame, write the prologue and epilogue for FuncQ. Don't use more instructions than you need to.

Prologue for FuncQ:

push %r12
push %rbp
~~sub $0x20, %rsp~~
sub $0x20, %rsp

Epilogue for FuncQ:

add %0x20, %rsp
pop %rbp
pop %r12

Stack Frames

	Arg. 8	FuncP
	Arg. 7	
	Return Address	
	Old r12	FuncQ
	Old rbp	
	Unused (16 bytes)	
rsp->	Space for C array int array[4]	

Part 2: FuncR is another function, and its assembly is shown below. Fill in the prologue and epilogue for this function too!

Prologue for FuncR:

push %rbx
sub $8, %rsp

Epilogue for FuncR:

add $8, %rsp
pop %rbx

```
___missing_FuncR_prologue___
movq     %rdi, %rbx
movq     $33, 8(%rsp)
movl     $1234, %esi
leaq     8(%rsp), %rdi
call     FuncS
addq     %rbx, %rax
___missing_FuncR_epilogue___
```

Question 6. Tricky Switch (7 pts)

Source Code

```
int func(int x, int y, int r) {
    switch (x) {
      case 0: ____Blank 0____
      case 1: ____Blank 1____
      case 2: ____Blank 2____
      case 3: ____Blank 3____
      case 4: ____Blank 4____
      case 5: ____Blank 5____
    }
    return ____Blank 6____;
}
```

Compiled Assembly

```
func(int, int, int):
        cmpl    $5, %edi
        ja      .L9   x≥5
        movl    %edi, %edi
        jmp     *.L4(,%rdi,8)
.L4:                               (hint, this is the
        .quad   .L8                 jump table!)
        .quad   .L7
        .quad   .L6
        .quad   .L5
        .quad   .L5
        .quad   .L3
.L7: case 1
        addl    $6, %edx    r+=6;
.L6: case 2
        leal    (%rdx,%rdx,4), %eax
        ret                            return 5r;
.L5: case 3, case 4
        leal    (%rdx,%rsi,2), %eax
        ret                            return 2y+r;
.L3: case 5
        movl    %edx, %eax   eax=r
        xorl    %esi, %eax   return r^y;
        ret
.L8: case 0
        movl    $-4, %eax
        Ret                  return -4;
.L9: x>5
        movl    %edx, %eax   return r;
        ret
```

Reverse engineer the assembly code on the previous page to figure out what each case of the switch-case statement is doing. Don't forget about "break" statements!

Blank 0	r-=4; break;
Blank 1	r+=6;
Blank 2	r*=5; break;
Blank 3	
Blank 4	r+=2y; break;
Blank 5	r=r^y; break;
Blank 6	r

Question 7. Floating Point Mystery (4pts)

A long time ago, we used to put floating-point questions on the datalab. I found a solution to one of these problems lying around, but can't figure out what it's doing anymore:

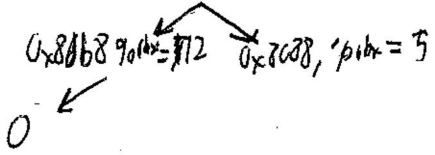
```
unsigned mystery_function(unsigned uf) {
  unsigned sign = uf>>31;      1 if negative, 0 if positive
  unsigned exp = uf>>23 & 0xFF; Selects out 8 exp bits
  unsigned frac = uf & 0x7FFFFF; Selects out 23 Frac bits
  if (exp == 0) { if denorm
    frac = 2*frac; scale frac <<1
    if (frac > 0x7FFFFF) { if overflow into exp
      frac = frac & 0x7FFFFF; ~~frac~~ select lower 22 frac bits with 23rd bit as 0
      exp = 1;    exp=1
    }
  } else if (exp < 0xFF) {If not inf
    exp++; increase exp by 1
    if (exp == 0xFF) { if inf condition
      frac = 0; make sure its a number
    }
  }
  return (sign << 31) | (exp << 23) | frac;
}
```

1. In what cases will this function return the same thing as the input argument? (1pt)

when uf = 0 or ~~[illegible]~~ exp = 0xFF

2. What does the above function do? (3pts)

The function multiplies the float represented by the bit pattern by 2.

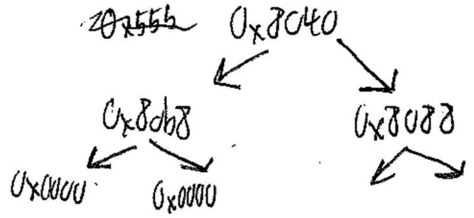


Question 8. Two Birds, One Bomb (6pts)

```
Dump of assembler code for function phase_5:
   0x00005555555551c8 <+0>:     sub    $0x8,%rsp
   0x00005555555551cc <+4>:     mov    $0xa,%edx   %edx=$0xa
   0x00005555555551d1 <+9>:     mov    $0x0,%esi   %esi = $0x0
   0x00005555555551d6 <+14>:    callq  0x555555555070 <strtol@plt>
   0x00005555555551db <+19>:    mov    %rax,%rsi   %rsi = first int
   0x00005555555551de <+22>:    lea    0x2e5b(%rip),%rdi        # 0x555555558040 <nodes>
   0x00005555555551e5 <+29>:    callq  0x55555555519e <recurse>
   0x00005555555551ea <+34>:    cmp    $0x21,%eax
   0x00005555555551ed <+37>:    je     0x5555555551fe <phase_5+54>
   0x00005555555551ef <+39>:    mov    $0x0,%eax
   0x00005555555551f4 <+44>:    callq  0x55555555517e <explode_bomb>
   0x00005555555551f9 <+49>:    add    $0x8,%rsp
   0x00005555555551fd <+53>:    retq
   0x00005555555551fe <+54>:    mov    $0x0,%eax
   0x0000555555555203 <+59>:    callq  0x555555555169 <phase_defused>
   0x0000555555555208 <+64>:    jmp    0x5555555551f9 <phase_5+49>
```

```
Dump of assembler code for function recurse:
   0x000055555555519e <+0>:     test   %rdi,%rdi   if rdi == 0
   0x00005555555551a1 <+3>:     je     0x5555555551c2 <recurse+36>
   0x00005555555551a3 <+5>:     push   %rbx
   0x00005555555551a4 <+6>:     mov    0x10(%rdi),%ebx   ebx = 0xa
   0x00005555555551a7 <+9>:     cmp    %esi,%ebx   if(0xa >= first int)       if (0x5 >= firstint)
   0x00005555555551a9 <+11>:    jge    0x5555555551b8 <recurse+26>
   0x00005555555551ab <+13>:    mov    0x8(%rdi),%rdi   rdi → value nextvalue
   0x00005555555551af <+17>:    callq  0x55555555519e <recurse>
   0x00005555555551b4 <+22>:    add    %ebx,%eax
   0x00005555555551b6 <+24>:    pop    %rbx
   0x00005555555551b7 <+25>:    retq
   0x00005555555551b8 <+26>:    mov    (%rdi),%rdi   rdi → value
   0x00005555555551bb <+29>:    callq  0x55555555519e <recurse>
   0x00005555555551c0 <+34>:    jmp    0x5555555551b4 <recurse+22>
   0x00005555555551c2 <+36>:    mov    $0x0,%eax
   0x00005555555551c7 <+41>:    retq
```

```
(gdb) x/21gx &nodes
0x555555558040 <nodes>:      0x0000555555558058  0x0000555555558070  0x000000000000000a   ebx = 10
0x555555558058 <nodes+24>:   0x0000555555558088  0x00005555555580a0  0x0000000000000005   ebx = 5
0x555555558070 <nodes+48>:   0x00005555555580b8  0x00005555555580d0  0x000000000000000c   ebx = 12
0x555555558088 <nodes+72>:   0x0000000000000000  0x0000000000000000  0x0000000000000001   ebx = 1
0x5555555580a0 <nodes+96>:   0x0000000000000000  0x0000000000000000  0x0000000000000007   ebx = 7
0x5555555580b8 <nodes+120>:  0x0000000000000000  0x0000000000000000  0x000000000000000b   ebx = 11
0x5555555580d0:<nodes+144>:  0x0000000000000000  0x0000000000000000  0x0000000000000011   ebx = 17
```



So, it turns out that I need a new phase_5 for the bomb lab, because the old phase_5 is waaaay too easy to cheat on ... therefore, I made this phase_5 for next year, with a similar kind of flavor.

Please help me make sure this new phase_5 has no errors by solving it for me. I've printed out a couple functions and some memory values. Next year's class will surely thank you. ;)

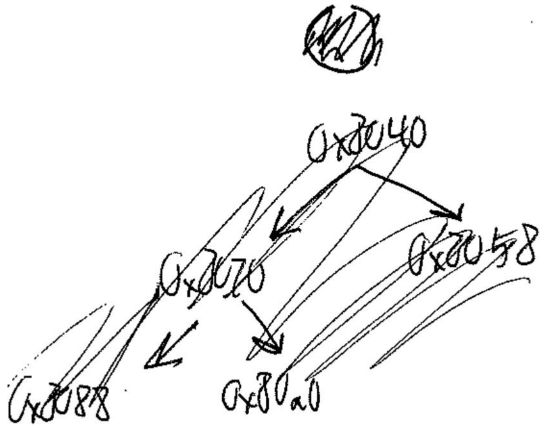
Note that "strtol" just converts an ascii string (specified by a char input) into an integer.*

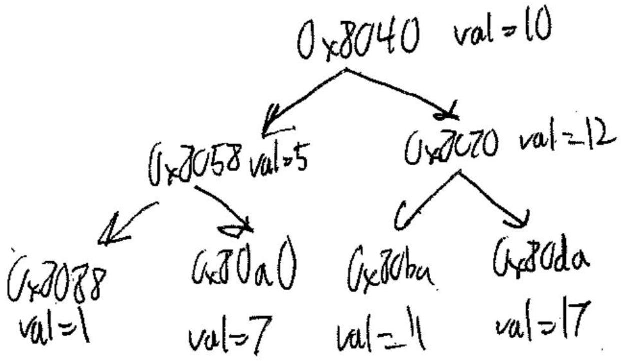
1. What datastructure is this problem concerned with? Please be specific. (1 pt)

This problem is concerned with a tree ~~[illegible], each with two~~ where each node points ~~[illegible]~~ to two other nodes and holds a valu (probably ~~[illegible]~~ a long)

2. What string passed to "phase_5(char* input)" will defuse the phase? (5 pts)

$\boxed{11}$





move right if input > val
move left if input ≤ val

11+12+10 = 33 = 0x21

Input 12 or 11 ✓