

**2021 Fall CPSC 240-3 Answers**  
**Final Concepts Test**  
**December 9, 2021 11:30am-3:00pm**

The policies are the same as in the previous tests. No need to repeat them here.

Write the word "Blank" in the answer space to obtain 20% of credit.

Send only one of these 3 formats: doc, docx, or odt. **Do not send PDF**

The total points for this test is 100.

Put your name and email address on this test on two different pages. You pick the pages.

Send to [holliday@fullerton.edu](mailto:holliday@fullerton.edu) PDF files will be deleted without warning.

End time = 3:00pm

1. Convert  $149\frac{3}{13}$  to IEEE754-32 bits hex. [12]

[Show enough intermediate steps to convince the grader that you really know 32-bit float numbers.]

Solution:

$149 = 10010101$  (by use of a calculator)

The fraction part presents a challenge. We will have to use repeated multiplication by 2 to find the repeating pattern.

$(3/13) \times 2$	$= 0 + (6/13)$	
$(6/13) \times 2$	$= 0 + (12/13)$	
$(12/13) \times 2$	$= 1 + (11/13)$	
$(11/13) \times 2$	$= 1 + (9/13)$	
$(9/13) \times 2$	$= 1 + (5/13)$	
$(5/13) \times 2$	$= 0 + (10/13)$	
$(10/13) \times 2$	$= 1 + (7/13)$	
$(7/13) \times 2$	$= 1 + (1/13)$	
$(1/13) \times 2$	$= 0 + (2/13)$	
$(2/13) \times 2$	$= 0 + (4/13)$	
$(4/13) \times 2$	$= 0 + (8/13)$	
$(8/13) \times 2$	$= 1 + (3/13)$	
$(3/13) \times 2$	$= 0 + (6/13)$	<== Repeating pattern begin here

Therefore, the original number is

$$10010101 . 001110110001 \overline{001110110001} \times 2^0$$

The sequence with overline repeats.

$$= 1 . 0010101001110110001 \overline{001110110001} \times 2^7$$

The true exponent is 7. Now add 7 and bias number.

The stored exponent is  $7F+7 = 8d = 1000\ 0110$

The significand is 23 bits repeating =  $00101010011101100010011$

Combine the parts: sign + stored exp + significand = IEEE formatted number

Our number =  $0\ 10000110\ 00101010011101100010011\ 1011$

The colors show groups of 4 bits per group.

Now write the number in IEEE format: **0x431A3B13 B**

The B on the right is outside the register, but since  $B \geq 8$  we round up the preceding digit.

Final answer ==> **0x431A3B14**

2. Convert 0x4589 6000 to a decimal float number. [12]

[Show enough intermediate steps to convince the grader that you really know 32-bit float numbers.]

Solution: First extract the stored exponent = 10001011 = 139 (decimal)

Subtract  $139 - 127 = 12$  (decimal) = true exponent

Next extract the significand from 896000 remembering to discard the left bit.

Hence, significand = 000 1001 0110 000 ... 0

Combine the part to obtain our number =  $1.000\ 1001\ 011 \times 2^{12}$

To simplify move the point 10 digits to the right.

Our number is  $10001001011. \times 2^2$

Using a calculator shows  $10001001011 = 1099$  (decimal)

Finally, our number is  $1099 \times 2^2 = 1099 \times 4 = 4396$

3. A software license (non-EULA) seeks to achieve certain outcomes for the end user. What outcomes are we speaking? [7]

Solution: Open source software licenses generally grant 4 freedoms (or rights) to the end consumer.

1. The right to receive the source code with the product itself.
2. The freedom distribute the software to others via any medium
3. The right to modify the program.
4. The freedom to distribute the modified product via any media.

4. What did Miss Ada invent that was novel in her day but is commonplace among programmers today? [7]

Solution

Ada invented the “call” feature of software.

She was first to manifest the concept of one software function calling another function and that called function would later surrender control back to the original function.

In her time period this was a technological breakthrough. Today we don't even pause to think about it.

When was her time? The professor did not look it up, but it had to be somewhere around 1840.

5. The next two numbers seem to be far apart. But are they really far apart? Show manually how a computer performs the process of adding the numbers. Show the sum. [12]

Have any data been lost in the process, and, if so, what is the numeric value of any lost data. Answers in decimal are preferred, but in hex will be considered.

All numbers are IEEE754-64 bits.

0x44A7 9000 0000 0000  
0x41BA 7000 0000 0000

Solution:

We'll give names to these two numbers

X = 0x44A7 9000 0000 0000

Y = 0x41BA 7000 0000 0000

For X the true exponent is  $0x44A - 0x3FF = 0x4B = 75$  (decimal)

For Y the true exponent is  $0x41B - 0x3FF = 0x1C = 28$  (decimal)


//The MATE calculator was used in the calculations above.

$X = 1.0111\ 1001\ 0000\ 0000\ \dots\ 0000 \times 2^{75}$

$Y = 1.1010\ 0111\ 0000\ 0000\ \dots\ 0000 \times 2^{28}$

We have to adjust the dot in Y to make its exponent agree with the exponent in X  
Move the dot in Y 47 places to the left.

$Y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 1010\ 0111 \times 2^{75}$



The underline shows the shift of 47 places.

Next line up the bits and add:

$X = 1.0111\ 1001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{75}$


$Y = 0.0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 1010\ 0111 \times 2^{75}$

The sum is

$X+Y = 1.0111\ 1001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 1010 \times 2^{75}$

$X+Y = 0x44A7\ 9000\ 0000\ 003A$

There is a loss of precision. The amount of loss is  
 $0.0000\ \dots\ 0000\ 0111 \times 2^{75}$



The underline represents an entire mantissa, namely: 52 bits of zeros.

In the loss number move the point 56 places to the right. Then the loss is  $7 \times 2^{19}$

That is sufficient. Use a calculator if you want a simpler expression.

6. The following dialog is part of the output of a larger program.

[13]

```
Please enter your name: Rudolf Red Nose
What is your precision number: 0.03.04.07
Invalid. Please try again: 7.4TR88
Invalid. Please try again: ++0.07
Invalid. Please try again: 0.01
Thank you Rudolf Red Nose. You're hired.
```

Show a portion of an X86 function that makes the dialog above. This is about validating an inputted double. You may use known functions from the C/C++ libraries. You may use publicly available functions. Do declare those functions as externs. Declare strings in the .data segment.

What if you run out of time? Scale it down. Perhaps omit the iteration. Maybe omit the part about Rudolf's name. You are an experienced programmer. Solving part of the problem is worth more points than solving none of it.

Solution:

This question should remind you of Assignment 2 (string I/O) and Assignment 4 (numeric input validation). The answer to this question combines parts from each of those two assignments.

The solution is long, but here it is.

```
null equ 0 ;null is the name of any sequence of zero bits
max_string_length equ 64
```

```
extern isfloat
extern scanf
extern printf
global get_precision
```

```
segment .data
prompt_for_name db "Please enter your name: ", 0
precision_question db "What is your precision number? ", 0
invalid_message db "Invalid. Please try again: ",0
departure db "Thank you %s. you're hired",10,0
```

```
segment .bss
namestring resb 64
```

```
segment .text
get_precision:
```

```
;Save time on a test: 15 pushes were omitted.
```

```
;Ask user for first and last names.
mov rax,0
mov rdi, prompt_for_name
call printf
```

```

;Obtain the user's names and store them in namestring
mov rax,0
mov rdi, namestring
mov rsi, max_str_length
mov rdx, [stdin]
call fgets

;Remove the newline character from the inputted string
mov rdi, namestring
call strlen
;The length is now in rax. Save a copy in r15, and then remove the newline
mov r15,rax
mov al,null
mov [namestring+r15-1], al ;al is the name of the lowest 1 byte of rax

;Prompt for the number of precision number
mov rdi, precision_question
call printf

;Make space for the incoming precision number
sub rsp,64 ;Create 64 bytes of space for the incoming precision number.

input_a_precision:
mov rdi, string_form
mov rsi, rsp
call scanf

;Is it a float?
mov rdi, rsp
call isfloat

;The response is in rax: rax == 0 means false; rax /= 0 means true
cmp rax,0
jne valid_input
mov rdi, invalid_message
call printf
jmp input_a_precision

valid_input:
mov rdi,rsp
call atof
;The precision number is now in xmm0. Save a backup copy in xmm15.
movsd xmm15,xmm0

;Say good-bye
mov rax,0
mov rdi,departure
mov rsi,namestring
call printf

;Set up the return value
mov xmm0, xmm15

add rsp,64 ;Reverse an earlier subtraction from rsp
;Fifteen pop were omitted.
ret

```

7. Here is a stack dump of the kind we all love.

[10]

Offset	Address	Value
+608	00007fff01d17e88	00007fff01d19271
+600	00007fff01d17e80	0000000000000000
+592	00007fff01d17e78	00007fff01d19269
+584	00007fff01d17e70	0000000000000001
+576	00007fff01d17e68	000000000000001c
+568	00007fff01d17e60	00007fff01d17e68
+560	00007fff01d17e58	00000000004004b9
+552	00007fff01d17e50	0000000000000000
+544	00007fff01d17e48	00007fff01d17e70
+536	00007fff01d17e40	0000000000000000
+528	00007fff01d17e38	00000008FF000000
+520	00007fff01d17e30	0000000000006678
+512	00007fff01d17e28	0000000000000001
+504	00007fff01d17e20	00007fff01d17e78
+496	00007fff01d17e18	0000000000401dd0
+488	00007fff01d17e10	00007fff01d17e40
+480	00007fff01d17e08	0000000000000000
+472	00007fff01d17e00	0000000000000000
+464	00007fff01d17df8	dda1284af90c3abc
+456	00007fff01d17df0	dc5cd74251bc3abc
+448	00007fff01d17de8	0000000000000000
+440	00007fff01d17de0	0000000000000000
+432	00007fff01d17dd8	00007fff01d17e70
+424	00007fff01d17dd0	0000000000400490
+416	00007fff01d17dc8	23a2d460915c3abc
+408	00007fff01d17dc0	0000000000000000
+400	00007fff01d17db8	0000000000400586
+392	00007fff01d17db0	0000000100000000
+384	00007fff01d17da8	00007fff01d17e78
+376	00007fff01d17da0	00007fff01d17e78
+368	00007fff01d17d98	00007f01fe552a40
+360	00007fff01d17d90	0000000000000000
+352	00007fff01d17d88	0000000000000020
+344	00007fff01d17d80	0000000000000006
+336	00007fff01d17d78	0000000000000005
+328	00007fff01d17d70	0000000000000004
+320	00007fff01d17d68	0000000000000003
+312	00007fff01d17d60	0000000000000002
+304	00007fff01d17d58	0000000000000001
+296	00007fff01d17d50	0000000000000001
+288	00007fff01d17d48	0000000100000000



+280	00007fff01d17d40	00007fff01d17e78
+272	00007fff01d17d38	00000000ffffffff
+264	00007fff01d17d30	0000000000400613
+256	00007fff01d17d28	00007fff01d17e10
+248	00007fff01d17d20	0000000000000008
+240	00007fff01d17d18	0000000000000007
+232	00007fff01d17d10	0000000000000006
+224	00007fff01d17d08	0000000000000002
+216	00007fff01d17d00	0000000000000002
+200	00007fff01d17cf0	00007fff01d17d30
+192	00007fff01d17ce8	0000000000000007
+184	00007fff01d17ce0	000000000000000a
+176	00007fff01d17cd8	0000000000000009
+168	00007fff01d17cd0	0000000000000000
+160	00007fff01d17cc8	0000000000000007
+152	00007fff01d17cc0	00007fff01d17e70
+136	00007fff01d17cb0	00007fff01d17cf0
+128	00007fff01d17ca8	0000000000000016
+120	00007fff01d17ca0	000000000000000b
+112	00007fff01d17c98	0000000000409ffc
+104	00007fff01d17c90	00007fff01d17d28
+96	00007fff01d17c88	00000000004007fc
+88	00007fff01d17c80	00007fff01d17cb0
+80	00007fff01d17c78	0000000000000012
+72	00007fff01d17c70	0000000000000000
+64	00007fff01d17c68	0000000000000012
+56	00007fff01d17c60	0000000000000000
+48	00007fff01d17c58	0000000000400876
+40	00007fff01d17c50	00007fff01d17c90
+32	00007fff01d17c48	000000000000001c
+24	00007fff01d17c40	00000000004020a7
+16	00007fff01d17c38	00007f01fe8f8970
+8	00007fff01d17c30	000000007fffffff
+0	00007fff01d17c28	ffffffffffffffff

Use your editor to place a blank line between the activation records.

The back end of each activation record in the linked list is highlighted in yellow.

8. Special technique. This is an IEEE754-32bit number. 0X7CD5 3000. It has a corresponding decimal expression, but that's not important here. [9]

How do you copy that 32-bit number into the lowest 32 bits of xmm9?

Solution:

Copy the literal number into the low half of a register like rax.

```
mov eax, 0X7CD5 3000
```

Push the entire register on the stack

```
push rax
```

Copy 32 bits from stack to lowest one-fourth of xmm9

```
movss xmm9, [rsp]
```

Return the stack to its former state

```
pop rax
```

9. This array has been declared in a C++ function: `char myname[12];` A cstring is already stored in the array. What is a gdb command that will show the ascii values of the chars in the array? [9]

Solution: `p/d myname`

10. What is a gdb command that will show the contents of the upper half of xmm3 in standard float format? [9]

Solution: `p/f xmm3.v2_double[1]`

End of test. Exactly 10 questions.

Put your name and email address on this test on two different pages.

Come and say hi in the CS building in the new semester.

Have a wonderful Christmas vacation.