University of Washington – Computer Science & Engineering

Winter 2021 | Instructor: Mark Wyse | Due: January 29, 2021

CSE 351 Study Guide 1 – Task 3

Last Name:	
First Name:	
UW NetID (username):	
Academic Integrity Statement: All work on these questions is my own. I had no prior knowledge of the questions, and I will not share or discuss my answers with anyone else. Violation of these terms may result in a failing grade. (**please sign**)	

Instructions

- Fill in your name and UW NetID in the table above, then read the Academic Integrity Statement and sign your name in the box to the right of it indicating that you understand and will comply with the statement.
- Show scratch work for partial credit but answer in the blanks, boxes, or spaces provided.
- You may use your study guide from Task 1, course lecture slides and Ed Lessons, and course textbooks while completing this task.
- Use of reference materials external to those listed above is not allowed (e.g., Stack Overflow, web searches, etc.)
- These questions should take approximately 30 minutes to answer.
- Refer to the Study Guides webpage for additional information: https://courses.cs.washington.edu/courses/cse351/21wi/guides/

Advice

- Read each question carefully.
- Relax and breathe; you are here to learn.

Part 1. Number Representation, Integers, Floating Point

(A) Assume that we are using w-bits to represent both two's complement and unsigned integers. Write the formulas to compute TMin, TMax, UMin, and UMax in the table below.

TMin	TMax	UMin	UMax

(B) In C, it is safe (there is no possibility of losing data) when casting from type `int` (4-byte integer) to type `float` (4-byte IEEE floating point). Circle your answer. No explanation required.

True / False

(C) The IEEE floating point standard allows any real number to be represented with both high precision and high accuracy. Circle your answer. No explanation required.

True / False

(D) Converting from type `int` to type `float` changes the stored bits. Circle your answer. No explanation required.

True / False

(E) Consider the table below where each row contains two 8-bit integral constants that will be compared using the **<**, **>**, or **==** comparison. Determine which comparison makes the expression: ***Left Constant* (<, >, ==) *Right Constant*** evaluate to ***True***. Also state the type of comparison that is performed (***signed*** or ***unsigned***) assuming we use the same type promotion and casting rules as C does.

Left Constant	Order (<, >, ==)	Right Constant	Comparison Type
1	**>**	**0**	**signed**
`(int) 15U`		`15`	
`(unsigned) -1`		`-2`	
`(unsigned) -128`		`127`	
`127`		`(int) 128U`	

(F) In one or two sentences, explain why floating-point operations do not work like real mathematics?

(G) In one or two sentences, explain the benefits of the Two's Complement integer representation compared to the Sign and Magnitude representation.

This space intentionally left blank.

Part 2. Memory, Data, and Addressing

(A) Assume we are executing code on a machine with a word size of w bits, and each addressable memory location stores b bytes. *What is the total size of the addressable memory on this machine?*

(B) A piece of data can be fetched from memory as long as we have its address. Circle your answer. No explanation required.

True / False

(C) Assume that a chunk of memory is located at address A and has size S. Write an equality comparison that evaluates to True when the memory chunk is aligned.

(D) How many bytes of memory does the following line of C code require on a 64-bit x86-64 architecture?

```
char *s = “I love CSE 351!”;
```

This space intentionally left blank.

(E) For this problem, assume we are executing on a 64-bit x86-64 machine (**little endian**) and that the initial contents of memory are shown below. The sizes of various C types are also listed below. Write the **type** and **hexadecimal value** for each expression in the table below, assuming these statements are evaluated after block of code listed below is executed. Write UNKNOWN if the value cannot be determined. Make sure to specify all bits for the result of each expression (i.e., use the correct number of bits as determined by the expression's resulting type).

Type	Size (in bytes)
char	1
short	2
int	4
long	8

Memory Address	+0	+1	+2	+3	+4	+5	+6	+7
0x00	30	31	32	33	34	35	36	00
0x08	FF	00	FF	00	AA	BB	CC	DD
0x10	AB	CD	EF	00	01	02	03	04
0x18	01	23	45	67	89	AB	CD	EF
0x20	BE	EF	BE	EF	BA	DD	BA	DD
0x28	11	22	33	44	55	66	77	88

```
char *c = 0x00;
short *s = (short*)(c + 32);
int *x = 0x10;
for (int i = 0; i < 3; i++) {
    x[i] = i*2;
}
long *y = (long*)(x + 4);
*((int*)(c+10)) = 0xFF00FF00;
```

Expression (in C)	Type	Value (hexadecimal)
c	char*	0x 0000 0000 0000 0000
y		
*(c+5)		
&y		
x[-1]		
(((short)x[2]) + 10)		
(short*)(((long*) &s[0]) - 2)		