Full Name:_______________________________________

Quiz I, Spring 2019 Date: 10/4

Instructions:

- Quiz I takes 70 minutes. Read through all the problems and complete the easy ones first.
- This exam is **closed book**, except that you may bring a single doube-sided page of prepared note.

1 (xx/24)	2 (xx/24)	3 (xx/31)	4 (xx/21)	Bonus (xx/8)	Total (xx/100)

This exam assumes 64-bit x86 hardware (little Endian) unless otherwise mentioned.

1 Machine representation, bitwise operation (24 points, 3 points each):

Answer the following multiple-choice questions. Circle *all* answers that apply.

A. Which value is the closest to 1¡¡20?

1. `1000`
2. `1000000`
3. `1 billion`
4. `2000`
5. `2000000`
6. `2 billion`

B. Given a 32-bit bit pattern `0xffffffff`, what is the value if we are to interpret the bit pattern as an unsigned int or a signed int?

1. 2^{31}
2. 2^{32}
3. $2^{31} - 1$
4. $2^{32} - 1$
5. -1
6. -2^{31}
7. -2^{32}
8. $-2^{31} + 1$
9. $-2^{32} + 1$
10. None of the above

C. Given a 32-bit bit pattern `0xffffffff`, what is the value if we are to interpret the bit pattern as an IEEE 32-bit floating point number?

1. NaN
2. inf
3. $-$ inf
4. 0
5. $\approx 2^{129}$
6. $\approx -2^{129}$
7. None of the above

D. Let `d1 = 123456789.2 - 123456789.1` and `d2=1.2-1.1`, what is the relationship of `d1` and `d2`?

1. `d1 == d2`
2. `d1 > d2`
3. `d1 < d2`
4. Any of the above.

E. Variable `x` is an `unsigned int` and `y` is an `unsigned char`. Which of the following statements assign `y` to contain the most significant byte of `x`?

1. `y = (unsigned char)x`
2. `y = (unsigned char)(x >> 24)`
3. `y = (unsigned char)(x | 0xff000000)`
4. `y = (unsigned char)(x & 0xff000000)`
5. None of the above

F. Which of the following expression evaluates to 0 *if and only if* the value of int variable `x` is 0?

1. `x & 0x00000000`
2. `x & 0xffffffff`
3. `x | 0xffffffff`
4. `x | 0x00000000`
5. `x & x`
6. None of the above

G. What is the output of the code snippet below (running on a Little-Endian machine)?

```
long long x = -2;
int *y;
y = (int *)&x;
printf("%d %d\n", y[0], y[1]);
```

1. -1 -1
2. -2 -2
3. -1 -2
4. -2 -1
5. Segmentation fault
6. None of the above

H. What is the output of the code snippet below (running on a Little-Endian machine)?

```
float f = 16.0;
unsigned int x;
x = (float *)&f;
x = x & 0x7fffffff;
printf("%f\n", *(float *)&x);
```

1. 16.0
2. -16.0
3. 0
4. some positive number
5. some negative number

2 Basic C (24 points, 3 points each)

Answer the following multiple-choice or fill-in-the-blank questions. Circle *all* answers that apply.

A. Given variable declaration `char *c[10];` what is the type of the expression `c[0]+1`?

1. `char**`
2. `char*`
3. `char`
4. `void*`
5. None of the above

B. Given variable declaration `char *c[10];` what is the type of the expression `c+1`?

1. `char**`
2. `char*`
3. `char`
4. `void*`
5. None of the above

C. Given variable declaration `char c[10];` what is the type of the expression `c[0]+1`

1. `char**`
2. `char*`
3. `char`
4. `void*`
5. None of the above

D. Given variable declaration `char c[10];` what is the type of the expression `c+1`

1. `char**`
2. `char*`
3. `char`
4. `void*`
5. None of the above

E. What is the output of the code snippet below?

```
    int a[2] = {1, 2};
    short *p;
    p = (short *)a;
    printf("%d %d %d\n", p[0], p[1], a[2]);
}
```

Answer: ____________________

F. What's the value of variable `p` after executing the statement `char p = '2' - 2;`? (ASCII Table is given in Appendix-A).

1. `'0'`
2. `0x30`
3. `0x31`
4. `0x32`
5. `'2'`
6. `'\0'`
7. `0x0`
8. None of the above

G. What is the output of running the following code snippet? (see Appendix-B for `strlen` manual)

```
char a[5] = {'a', 'b', 'c', 'd', '\0'};
printf("%d\n", strlen(a));
```

Answer: ____________________

H. What is the output of running the following code snippet?

```
char a[5] = {'a', 'b', 'c', 'd', '\0'};
char a[2] = '\0`;
printf("%d\n", strlen(a));
```

Answer: ____________________

3 C MiniLab (31 points):

In Lab1, you are asked to implement a function called `string_token`, to split a string into a sequence of tokens according to a specific delimiter character.

Each call to `string_token` returns a pointer to a null-terminated string containing the next token. A sequence of calls to `string_token` that operate on the same string maintains a pointer that determines the point from which to start searching for the next token. This pointer is saved in the variable pointed to by the `saveptr` argument.

On the first call to `string_token`, `str` should point to the string to be parsed, and the value of `saveptr` is ignored. In subsequent calls, `str` should be NULL, and `saveptr` should be unchanged since the previous call.

The code below is Ben Bitdiddle's implementation of `string_token` and his simple test.

```
char *
string_token(char *str, char delim, char **saveptr)
{
    if (!str)
        str = *saveptr;

    char *e;
    e = str;
    while ________________________________ {
        if ((*e) == delim) {
            *e = '\0';
            *saveptr = e+1;
            return str;
        }
        ____________________;
    }
    return NULL;
}
int main()
{
    char **saveptr;
    char test_str[10] = "2;12;13";
    char *token;
    token = string_token(test_str, ';', saveptr);
    while (token) {
       printf("[%s]\n", token);
       token = string_token(NULL, ';', saveptr);
    }
    printf("test_str is [%s]\n", test_str);
}
```

(a) (5 points) Assuming Ben's program is completed and works correctly, what is its expected output? (When answering this question, you can ignore the extra printf statement at line 35)?

(b) (8 points) Please complete line 9 and 15 in Ben's implementation of `string_token` to iterate through the string starting from the location pointed to by `e`.

(c) (6 points) When Ben actually runs his program, the dreaded "Segmentation fault" occurs. When he fires up `gdb`, he sees that the segmentation fault occurs at line 19. What is the reason for the segmentation fault?

1. It's because the type of the right handside expression `e+1` does not match that of the left handside `*saveptr` at line 12.
2. It's because `e+1` points to a location outside of the bound of the string `test_str`.
3. It's because line 11 is attempting to dereference an illegal address with the expression `*e`.
4. It's because line 12 is attempting to dereference an illegal address with the expression `*saveptr`.

(d) (6 points) Please fix Ben's program to elimininate the segmentation fault. You may only modify code in the `main` function and not elsewhere. (You can directly edit the code in the previous page)

(e) (6 points) What is the output of line 35?

(f) **Bonus question I: (8 points)** Suppose we replace line 22 of Ben's `main` program with the following two lines:

```
int x = 0x00413b42;
char *test_str = (char *)&x;
```

Assume Ben's program has been fixed as done in (d). What is the output of the program when running on a little-endian machine (You may ignore the printf at line 29)?

What is the output of the program when running on a big-endian machine (You may ignore the printf at line 29)?

4 More on C: Reversing a string (21 points)

(a) (5 points) Write the `swap` function that swaps two characters passed in by the caller.

(b) (8 points) Write the `reverse_str` function that reverses the characters in a given C string. This function modifies the original string in place. Your code **must** use the `swap` function that you've implemented in (a). You are free to use any of the string related functions in C library shown in Appendix-B.

```
void
reverse(char *str)
{



}
```

(c) (8 points) Complete the `main` function which tests the correctness of `reverse_str`. Specifically, you are to check that reversing a given string two times results in the string that's identical to the original copy. You are free to use any of the string related functions in C library shown in Appendix-B. **You code must have no more than 10 lines. Use** }

```
int
main()
{
   char *s;

   //get_rand_string returns a random string of a given length; it internally
   //performs malloc to allocate storage for the returned string.
   s = get_rand_string(7);

   //check that reversing s twice results in a string
   //that's identical to the original
















}
```

—END of Quiz I—-

Appendix A: ASCII

NAME

The following table contains the 128 ASCII characters encoded in octal, decimal, and hexadecimal

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL ’\0’	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL ’\a’ (bell)	107	71	47	G
010	8	08	BS ’\b’ (backspace)	110	72	48	H
011	9	09	HT ’\t’ (horizontal tab)	111	73	49	I
012	10	0A	LF ’\n’ (new line)	112	74	4A	J
013	11	0B	VT ’\v’ (vertical tab)	113	75	4B	K
014	12	0C	FF ’\f’ (form feed)	114	76	4C	L
015	13	0D	CR ’\r’ (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\ ’\\’
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27		147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	−	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

Appendix B: strlen, strncmp, strncpy in C library

```
STRLEN(3)              Linux Programmer's Manual

NAME
       strlen - calculate the length of a string

SYNOPSIS
       #include <string.h>

       size_t strlen(const char *s);

DESCRIPTION
       The strlen() function calculates the length of the string pointed to by s, excluding the terminating
       null byte ('\0').

RETURN VALUE
       The strlen() function returns the number of characters in the string pointed to by s.

-------------------------------------------------------------------------------------------------------------------
STRCMP(3)             Linux Programmer's Manual

NAME
       strcmp, strncmp - compare two strings

SYNOPSIS
       #include <string.h>

       int strcmp(const char *s1, const char *s2);

       int strncmp(const char *s1, const char *s2, size_t n);

DESCRIPTION
       The strcmp() function compares the two strings s1 and s2.  It returns an integer less than, equal to,
       or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

       The strncmp() function is similar, except it compares only the first (at most) n bytes of s1 and s2.

RETURN VALUE
       The strcmp() and strncmp() functions return an integer less than, equal to, or greater than zero if
       s1 (or the first n bytes thereof) is found, respectively, to be less than, to match, or be greater than s2

-------------------------------------------------------------------------------------------------------------------
STRCPY(3)           Linux Programmer's Manual

NAME
       strcpy, strncpy - copy a string

SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);

       char *strncpy(char *dest, const char *src, size_t n);

DESCRIPTION
       The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'),
       to the buffer pointed to by dest.  The strings may not overlap, and the destination string dest must be
       large enough to receive the copy.  Beware of buffer overruns!  (See BUGS.)

       The strncpy() function is similar, except that at most n bytes of src are copied.  Warning: If there is
       no null byte among the first n bytes of src, the string placed in dest will not be null-terminated.

       If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total
       of n bytes are written.
```