20F-COMSCI33-1 Midterm

TOTAL POINTS

39.5 / 60

QUESTION 1

Multiple Choice 12 pts

1.1 Binary vs other base? 2 / 2

+ **0 pts** Click here to replace this description.

✓ + **2 pts** **Click here to replace this description.**

+ **1.9 pts** format penalty

1.2 Data storage in address space 2 / 2

+ **0 pts** Click here to replace this description.

✓ + **2 pts** **Click here to replace this description.**

+ **1.9 pts** Format penalty

1.3 Implements “x * 2” 0 / 2

✓ + **0 pts** **Click here to replace this description.**

+ **2 pts** Click here to replace this description.

+ **1.9 pts** Format Penalty

1.4 Implements “x / 2”? 0 / 2

✓ + **0 pts** **Incorrect**

+ **2 pts** Click here to replace this description.

+ **1.9 pts** Format Penalty

1.5 Calling convention 0 / 2

✓ + **0 pts** **Click here to replace this description.**

+ **2 pts** Click here to replace this description.

+ **1.9 pts** Format Penalty

1.6 Value comparison 0 / 2

✓ + **0 pts** **Click here to replace this description.**

+ **2 pts** Click here to replace this description.

+ **1.9 pts** Format Penalty

QUESTION 2

Bit Manipulation 8 pts

2.1 func1 4 / 4

✓ + **4 pts** **Correctly states that func1 rotates a to the left by b, or states that func1 swaps two sections of bits in a separated by point b**

+ **1 pts** Alludes that a is left shifted by b bits and stored in P

+ **0 pts** Incorrect

2.2 func2 4 / 4

✓ + **4 pts** **Correctly states that func2 is the absolute value function**

+ **1 pts** Alludes that negative and positive values are treated differently

+ **0 pts** Incorrect

QUESTION 3

Novel Numbers 7 pts

3.1 1 - Binary Tmin 0.5 / 0.5

- **0.5 pts** Incorrect

✓ - **0 pts** **Correct**

3.2 1 - Decimal Tmin 0.5 / 0.5

✓ - **0 pts** **Correct**

- **0.5 pts** Wrong

3.3 1 - Binary Tmax 0.5 / 0.5

✓ - **0 pts** **Correct**

- **0.5 pts** Wrong

3.4 1 - Decimal Tmax 0.5 / 0.5

✓ - **0 pts** **Correct**

- **0.5 pts** Wrong

3.5 1 - Binary -1 0.5 / 0.5

✓ - **0 pts** **Correct**

- **0.5 pts** Wrong

3.6 1 - Binary -0 **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.7 1 - Binary +0 **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.8 2 - Binary Largest Normalized Number **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.9 2 - Decimal Largest Normalized Number **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.10 2 - Binary Smallest Positive Normalized Number **0 / 0.5**

- **0 pts** Correct

✓ - **0.5 pts Wrong**

3.11 2 - Decimal Smallest Positive Normalized Number **0 / 0.5**

- **0 pts** Correct

✓ - **0.5 pts Wrong**

3.12 2 - Binary -1 **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.13 2 - Binary -0 **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

3.14 2 - Binary +0 **0.5 / 0.5**

✓ - **0 pts Correct**

- **0.5 pts** Wrong

QUESTION 4

Pointy %rax 7 pts

4.1 addq1 **0 / 1**

- **0 pts** Correct

✓ - **1 pts Wrong**

4.2 addq2 **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

4.3 leaq1 **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

4.4 leaq2 **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

4.5 movq1 **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

4.6 movq2 **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

4.7 cmpq **1 / 1**

✓ - **0 pts Correct**

- **1 pts** Wrong

QUESTION 5

Struct and Union 10 pts

5.1 Struct overwatch **0 / 2**

✓ - **2 pts Incorrect**

- **0.1 pts** \-5% for bad formatting
- **0 pts** Correct
- **1 pts** Partial Credit

Correct Answer:
F F F F F F F F F F F F F F F P
first 8 bytes: tracer

next 4 bytes: mercy

next 2 bytes: slot3 (sizeof the union is 2 bytes)

next byte: brigite

last byte: padding (largest datatype is 8 bytes, need to pad up to the nearest multiple of 8 which in this case is 16)

5.2 Struct talon 2 / 2

✓ - **0 pts** Correct

- **2 pts** Click here to replace this description.
- **0.1 pts** 5% off for bad formatting
- **1 pts** Partial credit

5.3 GDB print 2 / 2

✓ - **0 pts** Correct

- **2 pts** Incorrect
- **1 pts** Incorrect Endianness
- **1 pts** Incorrect Output
- **0.1 pts** \-5% for bad formatting
- **0.05 pts** \-5% for bad formatting

5.4 Missing code 1 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.5 Missing code 2 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.6 Missing code 3 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.7 Missing code 4 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.8 Missing code 5 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.9 Missing code 6 0 / 0.5

- **0 pts** Correct

✓ - **0.5 pts** Incorrect

Answer: slot3.mei

5.10 Missing code 7 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

5.11 Missing code 8 0.5 / 0.5

✓ - **0 pts** Correct

- **0.5 pts** Incorrect

QUESTION 6

Stack 8 pts

6.1 Recursion 4 / 5

- **0 pts** Correct
- **1 pts** we know the value of rbx the second time we push it to stack
- **5 pts** Wrong
- **2 pts** wrong return addr
- **1 pts** extra fields
- **1.5 pts** specify return address
- **5 pts** missing
- **4 pts** partial
- **1.5 pts** wrong values of rbx

- 1 Point adjustment

6.2 Interpret func 3 / 3

✓ - **0 pts** Correct

- **3 pts** Missing
- **3 pts** Wrong
- **2 pts** Partial

QUESTION 7

Phantom 33 8 pts

7.1 Defuse 1 / 4

- **0 pts** Correct
- **1 pts** Slightly off / typo
- **1 pts** Base 10 instead of hex

✓ **- 3 pts** **On the right track, but incorrect**

- 4 pts Incorrect

7.2 s3cr3t **0 / 4**

- 0 pts Correct

- 1 pts Close but not correct

- 3 pts On the right track but incorrect

✓ **- 4 pts** **Incorrect**

Question 1. Multiple Choice (12 pts)

For the following multiple choice questions, select all that apply. If none of the answers are correct, simply leave the question blank. (2pts each, no partial credit)

1. Why do machines store information with binary (ie. base 2) instead of another base?
   a. Binary is more compact (eg. than decimal), so it saves memory space.
   b. Many circuit components are bistable, making it convenient for circuit design.
   c. Computer arithmetic is more efficient with a binary representation at the circuit level.
   d. Using higher bases makes it difficult to store numbers defined in lower bases.

2. What kind of data isn't stored within the address space of a program?
   a. Register Values
   b. Stack
   c. Heap
   d. Global Variables
   e. Program Binary

3. Suppose the variable "x" was defined as an "unsigned int" in C, and is stored in the "a" register (rax/eax/ax, etc.).
   Which of the following instructions correctly implements "x * 2"?

```
a. leal (%eax, %eax, 1), %eax
b. movl (%eax, %eax), %eax
c. addl (%eax), %eax
d. addl (,%eax, 1), %eax
e. addl %eax, %eax
f. sall 2, %eax
g. mulw 2, %ax
```

4. Suppose the variable "x" was defined as an "unsigned int" in C, and is stored in the "a" register (rax/eax/ax, etc.).
   Which of the following instructions correctly implements "x / 2"?

```
a. sall 2, %eax
b. sarl 2, %eax
c. sall 2, %eax (typo, yes it's the same as a)
d. sarl 1, %eax
e. divq 2, %rax
```

5. Which of the following registers are guaranteed to have a different value before and after a call instruction in x86-64?
    a. rax
    b. rbx
    c. rdi
    d. rbp
    e. rsp

6. Which of the following C statements are true?
    a. (8/5) == (8.0/5.0)
    b. (8/5) == (long) (8.0/5.0)
    c. (float) (8/5) == (8.0/5.0)
    d. (float) (8/5) == (long) (8.0/5.0)

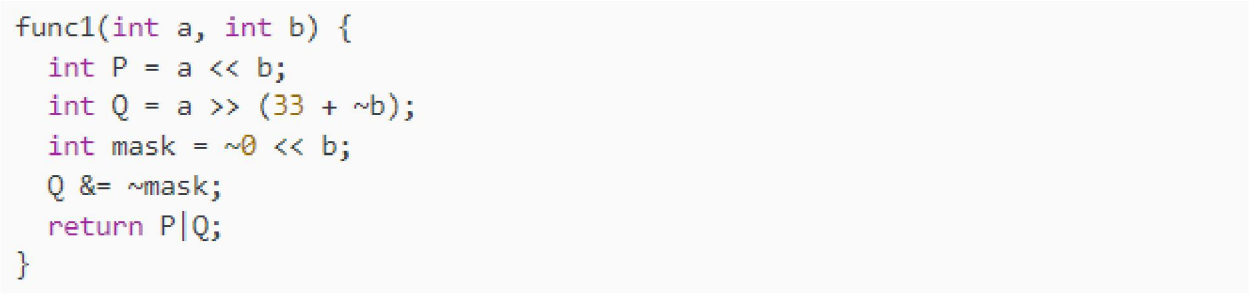
Multiple Choice Question Number	Write your answers here: (eg: a,b,d)
1.	B
2.	A
3.	A, B, D, E
4.	D, E
5.	A, C
6.	C, D

Question 2. A Bit of Manipulation (8 Pts)

Your friend gave you the solution to two of the datalab questions (nice friend!), but forgot to tell you which they were. Try to decipher them!

1. **func1 (4 Pts)**

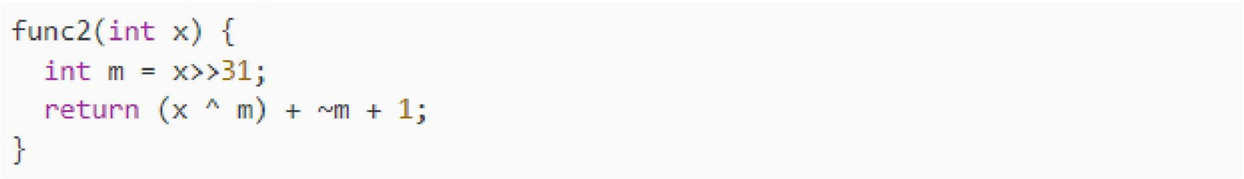
Hint: 1<=b<=31



```
func1(int a, int b) {
  int P = a << b;
  int Q = a >> (33 + ~b);
  int mask = ~0 << b;
  Q &= ~mask;
  return P|Q;
}
```

	Your answer in the cell below:
What does this function do? Please use only one or at most two sentences.	It performs a left circular shift on a by b bits, such that the most significant b bits of a get shifted to the

2. **func2 (4 Pts)**



```
func2(int x) {
  int m = x>>31;
  return (x ^ m) + ~m + 1;
}
```

	Your answer in the cell below:
What does this function do? Please use only one or at most two sentences.	The function returns the absolute value of x in 32 bits.

Question 3. Novel Numbers (7 pts)

Suppose we have a new machine where bytes are only 7 bits long, and there are no other datatypes. Luckily, we can still represent integer and floating point numbers easily.

1. Assuming standard two's complement representation, what are the following values: (assume 7-bit numbers)

	Binary	Decimal
Tmin	1000000	-64
Tmax	0111111	63
-1	1111111	
-0	0000000	
+0	0000000	

2. Assume we have a 7-bit floating point representation with 3 bits for the exponent, and otherwise we follow the normal floating point representation. (please remember that E=111 and E=000 is reserved for infinity/nan/denorm) What are the following values:

	Binary	Decimal
Largest Normalized Number	0110111	15
Smallest Positive Normalized Number	0001001	0.03125
-1	1011000	
-0	1000000	
+0	0000000	

Question 4. How pointy is your rax? (7 pts)

Based on each instruction individually, determine whether you think `%rax` is a pointer *before* the instruction is executed.

You have three options:
Yes -- There is evidence that `%rax` is a pointer.
No -- There is evidence that `%rax` is not a pointer.
Maybe -- There isn't evidence that `%rax` is a pointer or not a pointer.

	Is rax a pointer? (Options: Yes, No, Maybe)
`addq %rax, %rax`	Maybe
`addq %rbx, %rax`	Maybe
`leaq (%rbx, %rax, 4), %rcx`	No
`leaq (%rax, %rbx, 4), %rcx`	Maybe
`movq (%rbx, %rax, 4), %rcx`	No
`movq (%rax, %rbx, 4), %rcx`	Yes
`cmpq $5, %rax`	No

Question 5. Structures and Unions (10 pts)

Use the following structure definitions to answer the questions in this section.

```
struct overwatch {
    long* tracer;
    int mercy;
    union {
        char winston;
        short mei;
    } slot3;
    char brigite;
};

struct talon {
    int moira;
    short reaper;
    char sombra;
    char widowmaker;
};
```

1. Each cell in the following tables represents a byte. Each byte that is part of the struct can be part of a field (F) or padding (P). You need to fill out the table with letters (F or P) categorizing each byte. If a cell represents a byte that is not part of the data structure, leave it blank. (4pts)

`struct overwatch`

F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

`struct talon`

F	F	F	F	F	F	F	F								

2. Given the following output from gdb, what will be printed out by the last gdb command? (2pts)

```
(gdb) p buf
$1 = (unsigned char *) 0x8402260
(gdb) x/40xb buf
0x8402260:      0x67    0xc6    0x69    0x73    0x51    0xff    0x4a     0xec
0x8402268:      0x29    0xcd    0xba    0xab    0xf2    0xfb    0xe3     0x46
0x8402270:      0x7c    0xc2    0x54    0xf8    0x1b    0xe8    0xe7     0x8d
0x8402278:      0x76    0x5a    0x2e    0x63    0x33    0x9f    0xc9     0x9a
0x8402280:      0x66    0x32    0x0d    0xb7    0x31    0x58    0xa3     0x5a
(gdb) p/x ((struct overwatch*)buf)->slot3.mei
$2 = ……
```

What is printed:	`$2 = 0xfbf2`

3. Based on the following assembly code and incomplete C code. Please fill out the table with the missing C code that corresponds to the blanks in the C code. (4 pts)

```
00000000000005fa <capture_the_flag>:
5fa:   89 fe                   mov    %edi,%esi
5fc:   8b 05 36 0a 20 00       mov    0x200a36(%rip),%eax        # 201038 <overwatch+0x8>
602:   39 05 18 0a 20 00       cmp    %eax,0x200a18(%rip)        # 201020 <talon>
608:   0f 9f c1                setg   %cl
60b:   48 8d 05 12 0a 20 00    lea    0x200a12(%rip),%rax        # 201024 <talon+0x4>
612:   48 39 05 17 0a 20 00    cmp    %rax,0x200a17(%rip)        # 201030 <overwatch>
619:   76 36                   jbe    651 <capture_the_flag+0x57>
61b:   83 c9 80                or     $0xffffff80,%ecx
61e:   0f be 05 19 0a 20 00    movsbl 0x200a19(%rip),%eax        # 20103e <overwatch+0xe>
625:   0f bf 15 10 0a 20 00    movswl 0x200a10(%rip),%edx        # 20103c <overwatch+0xc>
62c:   01 d0                   add    %edx,%eax
62e:   0f be 15 f2 09 20 00    movsbl 0x2009f2(%rip),%edx        # 201027 <talon+0x7>
635:   0f be 3d ea 09 20 00    movsbl 0x2009ea(%rip),%edi        # 201026 <talon+0x6>
63c:   01 fa                   add    %edi,%edx
63e:   29 d0                   sub    %edx,%eax
640:   85 c0                   test   %eax,%eax
642:   7e 12                   jle    656 <capture_the_flag+0x5c>
644:   83 e6 7f                and    $0x7f,%esi
647:   40 38 ce                cmp    %cl,%sil
64a:   0f 9f c0                setg   %al
64d:   0f b6 c0                movzbl %al,%eax
650:   c3                      retq
651:   83 ce 80                or     $0xffffff80,%esi
654:   eb c8                   jmp    61e <capture_the_flag+0x24>
656:   83 e1 7f                and    $0x7f,%ecx
659:   eb ec                   jmp    647 <capture_the_flag+0x4d>
000000000000065b <main>:
65b:   bf 00 00 00 00          mov    $0x0,%edi
660:   e8 95 ff ff ff          callq  5fa <capture_the_flag>
665:   f3 c3                   repz retq
667:   66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
66e:   00 00
```

```
struct overwatch overwatch;
struct talon talon;
int capture_the_flag(char bias) {
   char winner = 0;
   if (talon.____1_____ > overwatch.____2____) { winner = 0x1; }
   if (overwatch.____3_____ > &talon.____4_____) { winner |= 0x80; }
   else { bias |= 0x80; }
   int overwatch_team = overwatch.____5_____ + overwatch.____6_____;
   int talon_team = talon.____7_____ + talon.____8____;
   if (overwatch_team - talon_team > 0) { bias &= 0x7f; } else { winner &= 0x7f; }
   return bias > winner;
}
int main() {
   return capture_the_flag(0x00);
}
```

Fill in your answers here:

Blank Number	Missing C Code
1	`moira`
2	`mercy`
3	`tracer`
4	`reaper`
5	`brigite`
6	`mei`
7	`widowmaker`
8	`sombra`

Question 6. Stack of Facts (8 pts)

Here is a recursive function: func(int x):

```
0000000000400b5d <func>:
  400b5d:       83 ff 01                cmp    $0x1,%edi
  400b60:       7f 06                   jg     400b68 <func+0xb>
  400b62:       b8 01 00 00 00          mov    $0x1,%eax
  400b67:       c3                      retq
  400b68:       53                      push   %rbx
  400b69:       89 fb                   mov    %edi,%ebx
  400b6b:       8d 7f ff                lea    -0x1(%rdi),%edi
  400b6e:       e8 ea ff ff ff          callq  400b5d <func>
  400b73:       0f af c3                imul   %ebx,%eax
  400b76:       5b                      pop    %rbx
  400b77:       c3                      retq
```

1. Suppose you call the recursive function func(3). Draw the stack when func(1) is entered. If you don't know a value, write "old" and then the value name. (eg. old %rax). (5pts)

[Return Address for Calling Function]
Old %rbx
0x400b73
2 (Value of %rbx)
0x400b73

(Assume each entry is 8 bytes, and don't use spaces you don't need!)

2. Figure out what this function is doing. (3pts)

What does this function do? (no more than one sentence)	It returns x! (x factorial)

Question 7. The Phantom 33 (8 pts)

Dear CS33: Attached is the final phase, removed from the bomblab because I couldn't solve it.

```
0000000000400b9c <get_magic_value>:
  400b9c:       48 8b 04 24             mov    (%rsp),%rax
  400ba0:       c3                      retq

0000000000400ba1 <phase_8>:
  400ba1:       53                      push   %rbx
  400ba2:       ba 10 00 00 00          mov    $0x10,%edx
  400ba7:       be 00 00 00 00          mov    $0x0,%esi
  400bac:       e8 7f e2 00 00          callq  40ee30 <__strtoul>
  400bb1:       48 89 c3                mov    %rax,%rbx
  400bb4:       b8 00 00 00 00          mov    $0x0,%eax
  400bb9:       e8 de ff ff ff          callq  400b9c <get_magic_value>
  400bbe:       48 39 d8                cmp    %rbx,%rax
  400bc1:       74 12                   je     400bd5 <phase_8+0x34>
  400bc3:       80 3c 18 21             cmpb   $0x21,(%rax,%rbx,1)
  400bc7:       74 18                   je     400be1 <phase_8+0x40>
  400bc9:       b8 00 00 00 00          mov    $0x0,%eax
  400bce:       e8 b4 ff ff ff          callq  400b87 <explode_bomb>
  400bd3:       5b                      pop    %rbx
  400bd4:       c3                      retq
  400bd5:       b8 00 00 00 00          mov    $0x0,%eax
  400bda:       e8 7e ff ff ff          callq  400b5d <phase_defused>
  400bdf:       eb f2                   jmp    400bd3 <phase_8+0x32>
  400be1:       b8 00 00 00 00          mov    $0x0,%eax
  400be6:       e8 87 ff ff ff          callq  400b72 <s3cr3t_phase>
  400beb:       eb e6                   jmp    400bd3 <phase_8+0x32>
```

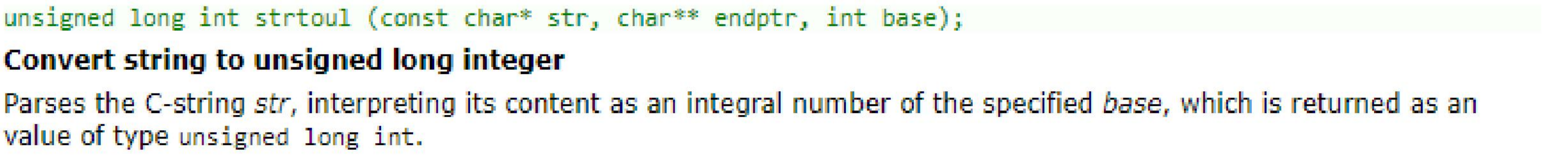
Also, I doubt this will be useful, but %rsp is 0x00676f7479610d0a when you enter phase_8.

Please let me know which input string will defuse this phase, and also how to find the secret phase. Return this table to me at your earliest convenience:

String to defuse:	gotya
String for s3cr3t:	ff63

Sincerely, Prof. Tony

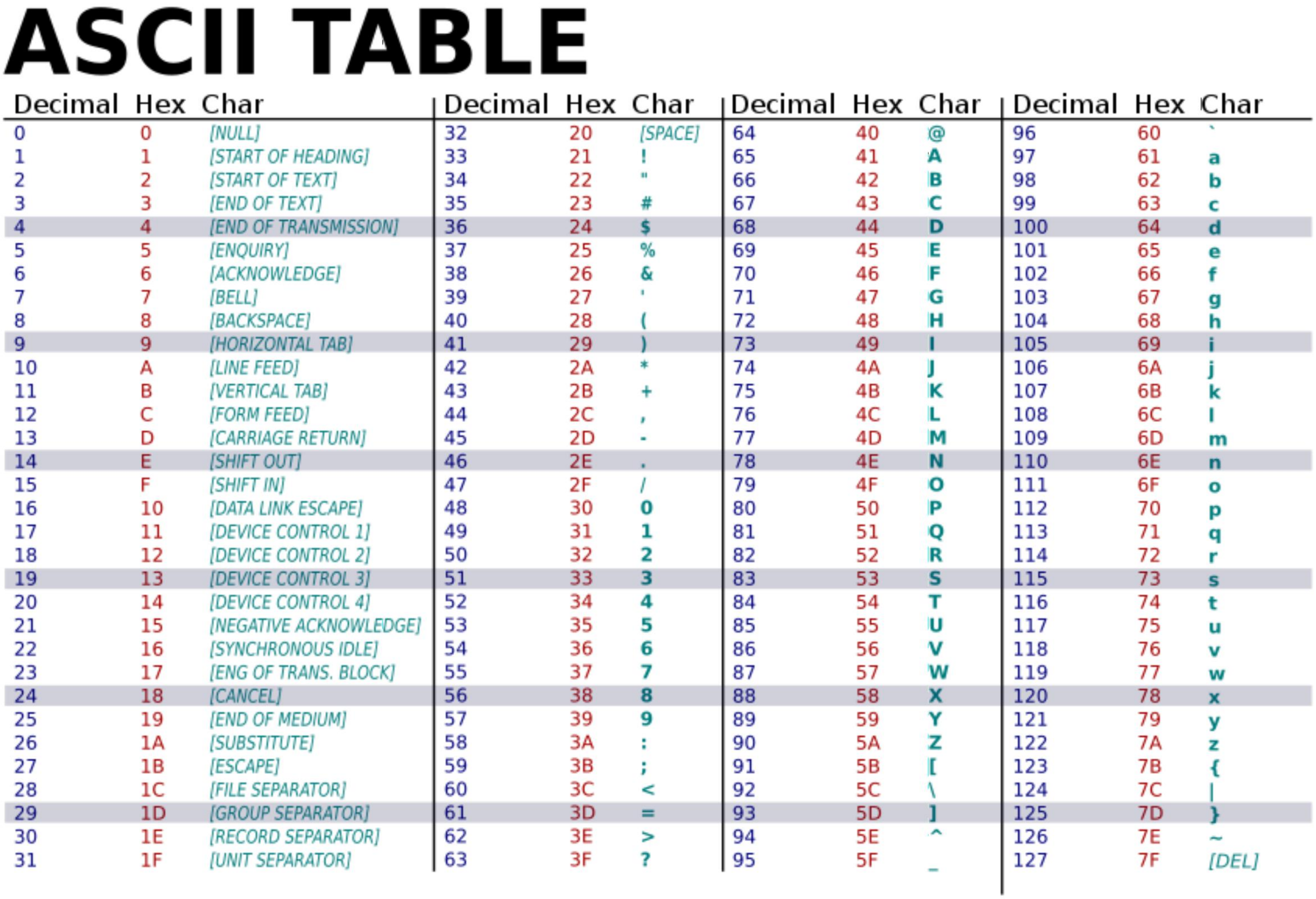
PS: I found this online, this actually might be useful.



```
unsigned long int strtoul (const char* str, char** endptr, int base);
```

Convert string to unsigned long integer

Parses the C-string *str*, interpreting its content as an integral number of the specified *base*, which is returned as an value of type `unsigned long int`.



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]