Vidya Akavoor
9/14/16

CAS CS 210 - Computer Systems
Fall 2016

PROBLEM SET 1 (PS1) ('C' BASICS, LOGIC AND DATA REPRESENTATION)
OUT: SEPTEMBER 13
DUE: SEPTEMBER 22, 1:30 PM

NO LATE SUBMISSIONS WILL BE ACCEPTED

To be completed individually. For all questions, show your work in obtaining the final answers.

PART A: 'C' Basics and Logic

1) hello.c

```
1 #include<stdio.h>
2
3 int
4 main(int argc, char **argv)
5 {
6   printf("Hello World\n");
7   return 0;
8 }
```

Describe the meaning/side effect of each non-blank line.

line 1 → imports the stdio.h library so that the program has access to the functions in that library
line 3 → indicates that the function will return an int and allocates enough memory to store an int.
line 4 → function header for function called main; int argc indicates the argument count; char **argv indicates a pointer to a pointer to an array of arguments; allocates 8 bytes of memory
line 5 → indicates beginning of program
line 6 → printf is a function that says to print/display the argument; will print "Hello World"; \n indicates to print a new line
line 7 → the return statement returns an int which matches what the header said it would (Line 3)
line 8 → indicates end of program

2) Memory and Pointers

```
1  #include <stdio.h>
2
3  char DNA[] = "ATCATCTC";
4  char *tptr;
5  int  tcnt;
6  int  len;
7
8  int process()
9  {
10   int found=0;
11
12   if (tptr == NULL) return 0;
13   while (*tptr) {
14     len++;
15     if (*tptr == 'T') {
16       tcnt++; found=1;
17     } else {
18       *tptr = 'a' + (*tptr - 'A');
19     }
20     tptr++;
21     if (found) return 1;
22   }
23   return 0;
24 }
25
26 int idx()
27 {
28   return (tptr) ? (tptr - DNA) : -1;
29 }
30
31 int main(int argc, char **argv)
32 {
33   tptr = DNA;
34   tcnt = 0;
35   len  = 0;
36
37   while (process()) printf("%d\n", idx());
38
39   printf("DNA:%s  tcnt:%d  len:%d\n", DNA, tcnt, len);
40   return 0;
41 }
```

tptr [DNA]
tcnt [0]
len [0]
found [0]

DNA: aTCATCTC
aTcATCTC
aTcaTCTC
aTcaTcTC
aTcaTcTc

1st time

2ND time

3RD time

len	tcnt	tptr
0	0	0x0000000000601037
1	0	35
2	1	36
3	1	37
4	1	38
5	2	39
6	2	3A
7	3	3B
8	3	3C

Please provide the output below for the program listing.

2
5
7

DNA: aTcaTcTc tcnt: 3 len: 8

Please fill in the missing values for the following table assuming that we stop the program just prior to it exiting at line 40. All address values should be written as 16 digit hex values and all integer values as simple decimals. chars should be treated as single byte integer values as such you should fill there decimal values in where appropriate. Consult the ascii table as necessary (eg. man ascii).

Name	Address	Value
DNA[0]	0x0000000000601034	97
DNA[1]	0x0000000000601035	84
DNA[2]	0x0000000000601036	99
DNA[3]	0x0000000000601037	97
DNA[4]	0x0000000000601038	84
DNA[5]	0x0000000000601039	99
DNA[6]	0x000000000060103a	84
DNA[7]	0x000000000060103b	99
DNA[8]	0x000000000060103c	0
tptr	0x0000000000601050	0x000000000060103C
tcnt	0x000000000060104c	3
len	0x0000000000601048	8

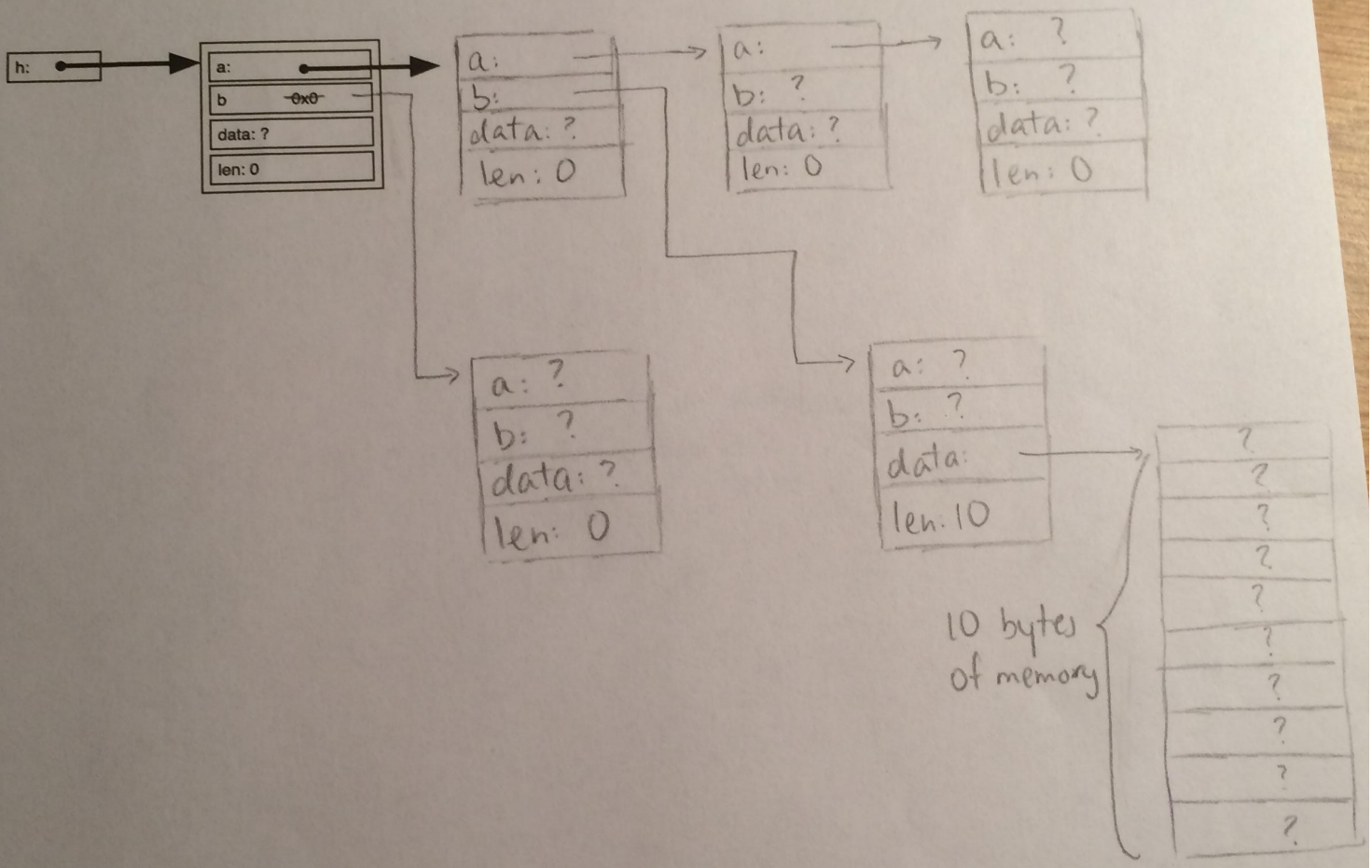
3) Pointers and Structs

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct myStruct{
5    struct myStruct *a;
6    struct myStruct *b;
7    char *data;
8    int len;
9  };
10
11   struct myStruct *h;
12
13 int main(){
14   h = malloc(sizeof(struct myStruct));
15   h->len = 0;
16   h->a = malloc(sizeof(struct myStruct));
17   h->a->len = 0;
18   h->a->a = malloc(sizeof(struct myStruct));
19   h->a->a->len = 0;
20   h->a->a->a = malloc(sizeof(struct myStruct));
21   h->a->a->a->len = 0;
22
23
24   h->b = malloc(sizeof(struct myStruct));
25   h->b->len = 0;
26   h->a->a->b = malloc(sizeof(struct myStruct));
27   h->a->a->b->len = 10;
28
29   h->a->a->b->data = malloc(10);
30
31   return 0;
32 }
```

A struct is a multi-byte programmer defined type that groups together several members (CBook ch 8). Sizeof can be used to determine the aggregate number of bytes that a particular struct type requires. Malloc is a standard 'C' library call that dynamically allocates the requested number of bytes of memory (CBook ch 16, p384-389). Malloc returns the address of the newly allocated memory. Storing the address in a variable of the appropriate pointer type allows you to access the memory allocated by malloc. In the case of a struct pointer you use the '->', called the member selection operator, to access a particular field of the struct pointed too. For further details on structs and malloc see the appropriate sections in the C book.

Complete the diagram on the next page. Illustrate the side effect of the above code fragment Draw all additional boxe and complete and add arrows as needed. Note assume there are no failure

in calls to malloc. You may use '?' to indicate unknown values of fields. Be sure to indicate all instances of the struct and all field values.



4) Logic Gates and Truth Tables)

1. Prove both of DeMorgan's Laws using Truth Tables. DeMorgan's Laws are:

   (a) Law 1: Stated in english and in 'C'

   English: Not A and B is the same as Not A or Not B

   'C': $!(A\&\&B) == (!A||!B)$

A	B	A && B	!(A && B)
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

!A	!B	!A || !B
1	1	1
0	1	1
1	0	1
0	0	0

Both have same truth values so

!(A&&B) = (!A || !B)

   (b) Law 2: Stated in english and in 'C'

   English: Not A or B is the same as Not A and Not B

   'C': $!(A||B) == (!A\&\&!B)$

A	B	A || B	!A || B
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

!A	!B	!A && !B
1	1	1
0	1	0
1	0	0
0	0	0

Both have same truth values so

!(A||B) = (!A && !B)

2. Only using NOT, OR and AND gates draw the logic gate cicuit for the following boolean expression. $(y||(z\&\&x))\&\&(!x||w)$