Spring 2017: CS 211: Final

Prof. Santosh Nagarakatte
May 9th 2017: 8am-11am

Full Name Here:

RUID:

NetID:

Instructions:
No electronic devices allowed.
Show your ID to the TA before you submit your exam.

Regular Credit: 100 points

Question	Max Points	Points
1	20	
2	20	
3	20	
4	20	
5	20	

Extra Credit: 30 points (7.5% towards the final grade)

Question	Max Points	Points
1	15	
2	15	

Problem 1: C Programming (20 points)

1. (5 points) You are implementing a hash table using chaining with linked lists where each node is of the following type.

```
struct node{
   char* key;
   char* value;
   struct node * next;
};
```

Given that the hash table is implemented as an array of pointers to hash table nodes, implement the following **hash_search** function to search a key in the hash table. If the key is found, the search function returns the `char*` value associated with the key and returns `NULL` otherwise. The value MAX_ENTRIES is the number of buckets in the hash table. Your code should carefully handle all corner cases, should compile, and should not experience segmentation faults on any input. You can assume that the hash function is already provided in the library and has the following prototype as shown below:

```
int hash_function(char*);
struct node * hash_table[MAX_ENTRIES];

char* hash_search(char* key){
```

2. (5 points) Write a **recursive** program named `reverse_list` to reverse a singly linked list pointed by pointer `list_ptr`. The function should return the pointer to the head of the reversed list.

```
struct node{
  void* data;
  struct node * next;
};

struct node* reverse_list(struct node * list_ptr) {
```

3. (5 points) You are given a binary search tree with each node having a left and a right child. If there are no children in the left or right branch, the respective pointers are NULL. Complete the `lookup` C function to look up a key in the binary search tree. The function returns 1 if the key is found and returns 0 if the key is not found.

```
 /* structure for the binary search tree */
struct node{
   int value;
   struct node* left;
   struct node* right;
};

 /* initially curr points to the root of the tree */
int lookup(struct node* root, int key){
```

4. (5 points) Write a C function to allocate a two-dimensional integer matrix with **m** rows and **n** columns. The use of the function is shown below.

```
int ** matrix;
...
matrix = allocate(p, q);
..
```

Your task is write a function that allocates the matrix and returns the appropriate pointer and fill the return type of the function.

```
  ....... allocate (int m, int n){
```

Problem 2: Data Representation + basic assembly

1. (5 points) You are given a 10-bit floating point representation with 1-sign bit, 3 bits for the exponent, and 6-bits for the mantissa. Express 16.875 in this representation. Clearly show work.

2. (5 points) Complete a C function below which checks if the *n*-th bit is 1 in the binary representation of a 64-bit number (size_t data type on a 64-bit machine) The function returns 1 if the n-th bit is 1 in the number and 0 otherwise. The argument specifying the *n*-th bit named *nbit* below ranges from 0 to 62. Hint: it is likely one line of code. Explain your intuition in 2 sentences. The datatype size_t is an unsigned 64-bit integer on 64-bit machines.

```
int check_bit(size_t number, int nbit){

  int condition =  ....


  return condition;

}
```

3. (4 points) You are creating a 6-bit representation of an integer.

   (a) What is the maximum value in two's complement 6-bit integer representation?

(b) What is the minimum value in two's complement 6-bit integer representation?

(c) What is the maximum value in one's complement 6-bit integer representation?

(d) What is the minimum value in one's complement 6-bit integer representation?

4. (3 points) You are given an unsigned **n**-bit number with value **x**. What is the value of the unsigned number obtained when you take the one's complement of number **x**? Show your work. Clearly state your intuition.

5. (3 points) Lets say we are designing a 7-bit floating point representation with 3-exponent bits, 1-sign bit and 3-mantissa bits.

   - What is the bias?
   - What is the largest normalized value?
   - What is the smallest normalized value?

Problem 3: Assembly (20 points)

1. (8 points) Write C code for the following assembly code.

```
foo:
     pushl   %ebp
     movl    %esp, %ebp
     movl    12(%ebp), %ecx
     addl    8(%ebp), %ecx
     movl    $1, %eax
     movl    $1, %edx
     cmpl    $1, %ecx
     jle     .L7
     testl   %ecx, %ecx
     jle     .L6
.L8:
     imull   %edx, %eax
     addl    $1, %edx
     cmpl    %edx, %ecx
     jge     .L8
     jmp     .L6
.L7:
.L6:
     popl    %ebp
     ret

.LC0:
     .string "%d  %d"
test:
     pushl   %ebp
     movl    %esp, %ebp
     subl    $40, %esp
     leal    -16(%ebp), %eax
     movl    %eax, 12(%esp)
     leal    -12(%ebp), %eax
     movl    %eax, 8(%esp)
     movl    $.LC0, 4(%esp)
     movl    8(%ebp), %eax
     movl    %eax, (%esp)
     call    sscanf
     movl    -16(%ebp), %eax
     movl    %eax, 4(%esp)
     movl    -12(%ebp), %eax
     movl    %eax, (%esp)
     call    foo
     leave
     ret
```

2. (2 points) What mathematical function does the above program compute? What is the result of the following function call?

```
result = test("5 1");
```

3. (10 points) As with the bomblab you have to devise the input to this program. There are multiple inputs that solve this phase named *foo*. **Identify all the inputs that would defuse this phase**. The function *explode_bomb* has the same behavior as in bomblab. The function *scanf* is the other function used in this program. This program can be diffused without using gdb. Read the question carefully. Each input to diffuse the bomb can require multiple integers. There are multiple distinct inputs that diffuse the bomb. Show your work.

```
.LC0:
        .string "%d"
        .text
.globl foo
        .type   foo, @function
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        leal    -12(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    scanf
        movl    -12(%ebp), %eax
        leal    1(%eax), %eax
        cmpl    $3, %eax
        je      .L3
        cmpl    $5, %eax
        jne     .L7
        jmp     .L8

.L3:
        leal    -16(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    scanf
        movl    -16(%ebp), %ecx
        leal    1(, %ecx,2), %ecx
        cmpl    $17, %ecx
        je      .L6
        call    explode_bomb
.L8:
        leal    -16(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    scanf
        cmpl    $19, -16(%ebp)
        je      .L6
.L7:
        call    explode_bomb
.L6:
        leave
        ret
```

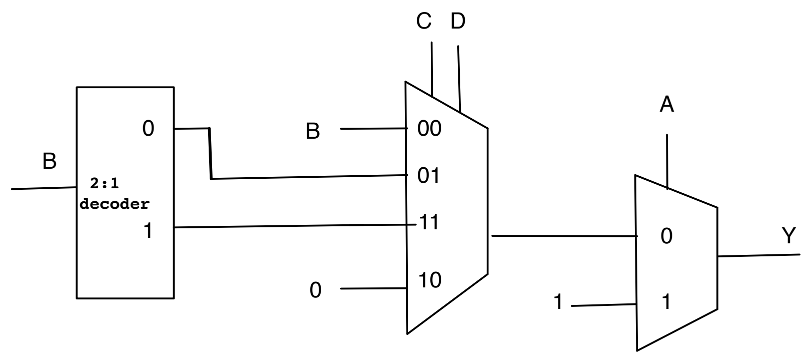

Figure 1: Logic Design Problem 1

Problem 4: Logic Design (20 points)

1. (5 points) Write the expression for the circuit in Figure 1. Subsequently simplify it using the K-map.

2. (5 points) Simplify the following boolean expression using Karnaugh maps.

   $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + AB\bar{C}D$

3. (10 points) Design an FSM that detects 1 0 1 in consecutive digits and outputs a 1 in the input stream of 0's and 1's. Implement the FSM using a combination of sequential and combinational logic. Clearly show your work. Draw the truth table for inputs, outputs and your states. Draw the K-map for everything. Clearly indicate how many flip flops you are going to use. Draw the final circuit with the flip flops.

```
INPUT :  0 1 0 1 0 1 1 0 1 0 0....
OUTPUT:  0 0 0 1 0 1 0 0 1 0 0....
```

/* Extra page for work here */

Problem 5: Caches (20 points)

1. Consider a machine with a direct mapped cache that has 8-sets and has a block size of 4-bytes. Memory addresses are 13-bits and each memory access from a processor requests a byte from the cache. The contents of the cache are as follows, with all numbers in hexadecimal.

	Cache Line					
Set Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	30	F4	72	AB
1	38	1	78	3F	92	D4
2	6E	0	-	-	-	-
3	06	0	1F	DA	40	C5
4	C7	1	-	-	-	-
5	71	1	D3	9A	0B	12
6	91	1	-	-	-	-
7	46	0	5C	80	B3	59

(a) (2 points) Calculate the size of the cache in bytes.

(b) (3 points) Indicate the number of bits used to determine

- the block offset:
- set index:
- tag:

(c) (10 points) A program running on this machine makes memory references at the addresses specified in the first column of the following table. For each memory reference indicate what is the block offset, set index and tag. Furthermore, indicate whether a cache hit occurs and the byte returned. The byte returned is - for a miss.

Address	Set index	Block offset	Tag	Cache hit? (Y/N)	Byte returned
0x0E35					
0x0DD5					
0x1FE4					
0x0705					

(d) (3 points) If we redesign the cache by increasing the associativity to 4 while keeping the total cache size exactly the same as above, how many bits will you use for the following:

- Tag:
- Set index:
- Block offset:

(e) (2 points) What is spatial locality and temporal locality?

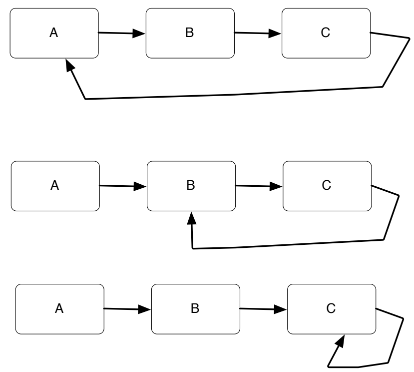

Figure 2: Examples of cycles for a linked list with 3 nodes.

Extra Credit Questions

Extra Credit 1: C Programming (15 points)

You are given a pointer to a singly linked list. The singly linked list has cycles due to a programming error. **Write a C program to detect whether the linked list has cycles without storing all the pointers to the nodes**. The function `detect_cycles` should return 1 when a cycle is found and 0 when there are no cycles. Your code should handle all corner cases carefully and should not cause segmentation fault.

Figure shows various examples of cycles for a list with 3 nodes. The number of nodes in the input list can range from 0 to a billion nodes.

```
struct node{
  void* data;
  struct node* next;
};

int detect_cycles(struct node* list){
```

/* another page for work */

Extra Credit 2: Assembly (15 points)

You are charged with maintaining a large C program and you come across the following code.

```
typedef struct{
    int left;
    a_struct a[CNT];
    int right;
} b_struct;

void test(int i, b_struct *bp){

   int n = bp->left + bp->right;
   a_struct* ap = &bp->a[i];
   ap->x[ap->idx] = n;
}
```

The declaration of the compile time constant CNT and the structure a_struct are in a file for which you don't have necessary access privilege. Fortunately you have a copy of the '.o' version of code, which you are able to disassemble with objdump program yielding the disassembly shown below.

```
test:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        imull   $28, %edx, %ecx
        leal    (%eax,%ecx), %ecx
        leal    0(,%edx,8), %ebx
        subl    %edx, %ebx
        movl    %ebx, %edx
        addl    4(%ecx), %edx
        movl    312(%eax), %ebx
        addl    (%eax), %ebx
        addl    8(%ecx), %ebx
        movl    %ebx, 12(%eax,%edx,4)
        popl    %ebx
        popl    %ebp
        ret
```

Clearly show your reasoning for your answers. Using your reverse engineering skills, deduce the following:

1. The value of CNT.

2. A complete declaration of structure a_struct.

/* show work here */