You may use one page of hand written notes (both sides) and a dictionary.
No i-phones, calculators nor any other type of non-organic computer.

1) Linked List (version 1): This C program compiles and runs. It correctly adds char arrays (names) to a linked list in lexical order. What is its output?

syntax of **strcmp** from **string.h**

```
int strcmp ( const char * str1, const char * str2 )
```

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.
Returns: zero if both strings are equal. A value greater than zero if the first character that does not match has a greater value in str1 than in str2. Otherwise, returns a value less than zero.

```
 1) #include <stdio.h>
 2) #include <stdlib.h>
 3) #include <string.h>
 4)
 5) #define MAX_NAME_LEN 16
 6)
 7) struct Node
 8) {
 9)   struct Node *next;
10)   char name[MAX_NAME_LEN];
11) };
12)
13) struct Node *start = NULL;
14)
15) void addNode(char* name);
16)
17) //=======================================================
18) void main(void)
19) //=======================================================
20) {
21)   addNode("something");
22)   addNode("wicked");
23)   addNode("this");
24)   addNode("way");
25)   addNode("come");
26) }
27)
```

```
28) //=====================================================
29) void addNode(char* name)
30) //=====================================================
31) {
32)   struct Node *newNode = malloc(sizeof(struct Node));
33)   strcpy(newNode->name, name);  //string copy
34)
35)   if(start == NULL)
36)   { printf("addNode(%s) NULL", name);
37)     start = newNode;
38)     start->next = NULL;
39)   }
40)
41)   else if (strcmp(start->name, name) > 0)
42)   { printf("addNode(%s) START",name);
43)     newNode->next = start;
44)     start = newNode;
45)   }
46)   else
47)   {
48)     printf("addNode(%s) ", name);
49)     struct Node *node = start;
50)     int done = 0;
51)     while(!done)
52)     {
53)       printf("%s ", node->name);
54)       if (node->next == NULL)
55)       { node->next = newNode;
56)         newNode->next = NULL;
57)         done = 1;
58)       }
59)
60)       else if (strcmp(node->next->name, name) > 0)
61)       { newNode->next = node->next;
62)         node->next = newNode;
63)         done = 1;
64)       }
65)
66)       node = node->next;
67)     }
68)   }
69)   printf("\n");
70) }
71)
```

2) Linked List (version 2): The removeNode function below is added to linked list version 1 and the main function in version 1 is replaced with the main shown below. This version 2 of linked list compiles and runs. However, it segfaults.

Which is the first line (in execution order) on which a segfault might occur?

Fix the removeNode function so that it correctly removes nodes.

```
 1) //=========================================================
 2) void removeNode(char* name)
 3) //=========================================================
 4) {
 5)   struct Node* deletedNode = NULL;
 6)
 7)   if (start == NULL) return;
 8)   if (strcmp(start->name, name) == 0)
 9)   {
10)     deletedNode = start;
11)     free(deletedNode);
12)   }
13)
14)   else
15)   {
16)     struct Node *node;
17)     node = start;
18)     while(node->next)
19)     {
20)       if (strcmp(node->next->name, name) == 0)
21)       {
22)         deletedNode = node->next;
23)         free(deletedNode);
24)       }
25)       node = node->next;
26)     }
27)   }
28) }
29)
30)
31)
32)
33) //=========================================================
34) void main(void)
35) //=========================================================
36) {
37)   addNode("something");
```

```
38)   addNode("wicked");
39)   addNode("this");
40)   addNode("way");
41)   addNode("come");
42)   removeNode("way");
43)   removeNode("macbeth");
44)   removeNode("something");
45)   while(start) removeNode(start->name);
46) }
```

3) Bit Operators: This C program compiles and runs. What is its output?

```
 1) #include <stdio.h>
 2) void main(void)
 3) { unsigned char x = 103;
 4)
 5)   unsigned char a = x << 3;
 6)   unsigned char b = x >> 3;
 7)   unsigned char c = x & 10;
 8)   unsigned char d = x & 59;
 9)   unsigned char e = x | 10;
10)   unsigned char f = x ^ 10;;
11)
12)   printf("a=%d, b=%d, c=%d, d=%d, e=%d, f=%d\n",
13)           a, b, c, d, e, f);
14) }
```

4) This C program compiles and runs. If the output from lines 7 and 8 is:

```
sizeof(long)=8
x=0x7fff29af6530, x[0]=22
```

Then what is the output from line 10?

```
 1) #include <stdio.h>
 2)
 3) void main(void)
 4) {
 5)   long a[] = {22, 33, 44, 55, 66, 77};
 6)   long *x = a;
 7)   printf("sizeof(long)=%lu\n", sizeof(long));
 8)   printf("x=%p, x[0]=%d\n", x, x[0]);
 9)   x = x + 3;
10)   printf("x=%p, x[0]=%d\n", x, x[0]);
11) }
```

5) This C program compiles and runs. What is the output?

```
 1)  #include <stdio.h>
 2)  int level;
 3)
 4)  void swap(int v[], int i, int j)
 5)  { int c = v[i];
 6)    v[i] = v[j];
 7)    v[j] = c;
 8)  }
 9)
10)  void quicksort(int v[], int left, int right)
11)  {
12)    level++;
13)    printf("level=%d: %d, %d\n", level, left, right);
14)
15)    int i, last;
16)    if (left < right)
17)    {
18)      swap(v, left, (left+right)/2);
19)      last = left;
20)      int num2 = v[left];
21)      for (i=left+1; i <= right; i++)
22)      {
23)        if (v[i] < v[left])
24)        { last++;
25)          swap(v, last, i);
26)        }
27)      }
28)
29)      swap(v, left, last);
30)      quicksort(v, left, last-1);
31)      quicksort(v, last+1, right);
32)    }
33)    level--;
34)  }
35)
36)
37)  void main(void)
38)  {
39)    int v[] = {55, 22, 77, 88, 33, 11};
40)    int arraySize = sizeof(v)/sizeof(int);
41)    level = 0;
42)    quicksort(v, 0, arraySize-1);
43)  }
```