19.1 Is there a bug in the following program? Explain.

```c
struct node {
    int count;
    struct node *next;
};

int main()
{
    int data = 0;
    struct node *getdata;

    getdata->count = data + 1;
    printf("%d", getdata->count);
}
```

19.2 The following are a few lines of a C program:

```c
struct node {
    int count;
    struct node *next;
};

main()
{
    int data = 0;
    struct node *getdata;

    :
    :

    getdata = getdata->next;
    :
    :
}
```

Write, in LC-3 assembly language, the instructions that are generated by the compiler for the line `getdata = getdata->next;`.
19.3 The code for PotentialCollisions in Figure 19.2 performs a pairwise check of all aircraft currently in the airspace. It checks each plane with every other plane for a potential collision scenario. This code, however, can be made more efficient with a very simple change. What is the change?

19.4 The following program is compiled on a machine in which each basic data type (pointer, character, integer, floating point) occupies one location of memory.

```c
struct element {
    char name[25];
    int atomic_number;
    float atomic_mass;
};

int is_it_noble(struct element t[], int i) {
    if ((t[i].atomic_number==2) ||
        (t[i].atomic_number==10) ||
        (t[i].atomic_number==18) ||
        (t[i].atomic_number==36) ||
        (t[i].atomic_number==54) ||
        (t[i].atomic_number==86))
        return 1;
    else
        return 0;
}

int main() {
    int x, y;
    struct element periodic_table[110];

    x = is_it_noble(periodic_table, y);
    ...}
```

a. How many locations will the activation record of the function is_it_noble contain?

b. Assuming that periodic_table, x, and y are the only local variables, how many locations in the activation record for main will be devoted to local variables?
19.5 The following C program is compiled into the LC-3 machine language and executed. The run-time stack begins at xEFFF. The user types the input abac followed by a return.

```c
#include <stdio.h>
#define MAX 4

struct char_rec {
    char ch;
    struct char_rec *back;
};

int main()
{
    struct char_rec *ptr, pat[MAX+2];
    int i = 1, j = 1;
    printf("Pattern: ");
    pat[1].back = pat;
    ptr = pat;
    while ((pat[i].ch = getchar()) != '\n') {
        ptr[++i].back = ++ptr;
        if (i > MAX) break;
    }
    while (j <= i)
        printf("%d ", pat[j++].back - pat);
/* Note the pointer arithmetic here: subtraction of pointers to structures gives the number of structures between addresses, not the number of memory locations */
}
```

a. Show the contents of the activation record for `main` when the program terminates.
b. What is the output of this program for the input `abac`?