(i) (6 pts.)

Consider the following C program, which includes three function definitions, including the **`main`** function shown on the next page.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define LIMIT 3
#define DISPLAY_LENGTH 8

struct element {
    char text;  struct element *next;
};

struct element *add_elements(struct element *h, char c1, char c2) {
    struct element *t1 = NULL;
    struct element *t2 = NULL;
    struct element *tmp = NULL;

    t1 = malloc(sizeof(struct element));
    t2 = malloc(sizeof(struct element));
    t1->text = c1;
    t2->text = c2;
    if (isdigit(c1)) {
        tmp = t2;
        t2 = t1;
        t1 = tmp;
    }
    t1->next = t2;
    if (h == NULL) {
        t2->next = t1;
    } else {
        t2->next = h->next;
        h->next = t1;
    }
    return t1;
}

void print_elements(struct element *elements, int num) {
    int i = 0;
    struct element *p = NULL;

    p = elements;
    for (i = 0; i < num; i++) {
        printf("item %d: %c\n", i, p->text);
        p = p->next;
    }
}
```

```
int main(void) {
    struct element *data = NULL;
    char *name1 = "M1a3y";
    char *name2 = "J5u7n9e";
    int i = 0;
    int wrap = 6;

    for (i = 0; i < LIMIT; i++) {
        data = add_elements(data, name1[i], name2[i % wrap]);
    }
    print_elements(data, DISPLAY_LENGTH);
    return 0;
}
```

Show the complete output as it appears on standard output. Show all work, and clearly indicate your solution. Show your work and your solution for this problem *only* on the *previous* page and (if more space is needed) *this* page.

In case of an illegal dereferencing of a pointer (e.g., dereferencing of an uninitialized pointer, null pointer, or pointer that goes beyond the boundaries of an array), show all of the output from the **`printf`** calls that are executed up to the point just before the illegal pointer dereference, and then write "illegal pointer operation" on the following line.

(ii) (4 pts.)

Consider the following C program.

```
#include <stdio.h>
#include <stdlib.h>

#define N1 10
#define N2 5
#define CH 's'

int main(void) {
    int i = 0;
    int iter = 0;
    int index = 1;
    char s[] = "test-string";

    for (i = 1; i < N1; i++) {
        iter++;
        i += N1 / (i + 1);
    }
    printf("iter #%d = %d\n", index, iter);

    iter = 0;
    index++;
    while (i > 0) {
        iter++;
        i -= (N2 % iter);
    }
    printf("iter #%d = %d\n", index, iter);

    iter = 0;
    i = N2;
    index++;
    do {
        iter++;
        i = (i + N2 + 1) % N1;
    } while (s[i] != CH);
    printf("iter #%d = %d\n", index, iter);

    return 0;
}
```

Show the complete output as it appears on standard output. Show all work, and clearly indicate your solution. Show your work and your solution for this problem *only* on *this* page and (if more space is needed) *the next* page.

(iii) (6 pts.)

Consider the following C program, which includes three function definitions, including the **`main`** function.

```
#include <stdio.h>

void dump_values(int v1, int v2, int v3, int v4) {
    printf("v1 = %d, v2 = %d, v3 = %d, v4 = %d\n",
            v1, v2, v3, v4);
}

int f(int *x, int y) {
    printf("Calling with %d and %d.\n", *x, y);
    if ((*x) < y) {
        return f(&y, (*x)) + 1;
    } else {
        y = (*x) + 2;
        (*x) = y + 3;
        return (*x) + y;
    }
}

int main(void) {
    int result = 0;
    int a = 0;
    int b = 0;
    int c = 100;

    result = f(&a, b);
    dump_values(a, b, c, result);
    result = f(&b, a);
    dump_values(a, b, c, result);
    result = f(&a, f(&b, c));
    dump_values(a, b, c, result);

    return 0;
}
```

Show the complete output as it appears on standard output. Show all work, and clearly indicate your solution. Show your work and your solution for this problem *only* on *this* page and (if more space is needed) *the next* page.

In case of an illegal dereferencing of a pointer (e.g., dereferencing of an uninitialized pointer, null pointer, or pointer that goes beyond the boundaries of an array), show all of the output from the **`printf`** calls that are executed up to the point just before the illegal pointer dereference, and then write "illegal pointer operation" on the following line.

(iv) (4 pts.)

Consider the following structure definition for constructing linked lists of records, where each record holds the first and last name of a single person.

```
struct person_entry {
    /* First and last names */
    char *first_name;
    char *last_name;

    /* Pointer to the next list element */
    struct person_entry *next;
};
```

Consider also the following function prototype and associated header comment.

```
/*****************************************************************
Given a list of records in some database of people, pointed to by the
"head" argument, determine if there is a record ("matching record") in
the list whose first and last names match the "fname" and "lname"
arguments, respectively. If there is exactly one matching record, then
remove that matching record from the list and return a pointer to the
matching record.  If there are multiple matching records, then remove
the first matching record from the list and return a pointer to the
removed matching record. If there is no matching record, then return a
null pointer.  Assume that the first element in the list (the element
pointed to by the "head" argument) is a "dummy" record whose purpose
is only to mark the beginning of the list, and assume that this dummy
record is not a matching record.
*****************************************************************/
struct person_entry *delete_record(struct person_entry *head,
        char *fname, char *lname);
```

Note that no error checking is required in this function beyond what is stated in the header comment. For example, it can be assumed that the given "head" pointer is non-NULL.

Develop a complete C code implementation of the function **`delete_record`**.

Show all work, and clearly indicate your solution. Show your work and your solution for this problem *only* on the *this* page and (if more space is needed) *the next* page

Software Qualifying Exam Solutions

Fall 2016
Dept. of ECE, University of Maryland, College Park
9/9/2016

Problem 1:

```
item 0: a
item 1: u
item 2: 1
item 3: J
item 4: M
item 5: 5
item 6: a
item 7: u
```

Problem 2:

```
iter #1 = 3
iter #2 = 7
iter #3 = 5
```

Problem 3:

```
Calling with 0 and 0.
v1 = 5, v2 = 0, v3 = 100, v4 = 7
Calling with 0 and 5.
Calling with 5 and 0.
v1 = 5, v2 = 0, v3 = 100, v4 = 18
Calling with 0 and 100.
Calling with 100 and 0.
Calling with 5 and 208.
Calling with 208 and 5.
v1 = 5, v2 = 0, v3 = 100, v4 = 424
```

Problem 4:

```
struct person_entry *delete_record(struct person_entry *head,
        char *fname, char *lname) {
    struct person_entry *p = NULL;
    struct person_entry *match = NULL;
    int found = FALSE;

    p = head;
    while ((p != NULL) && (!found)) {
        found = ((strcmp(p->first_name, fname) == 0) &&
                (strcmp(p->last_name, lname) == 0));
        if (!found) {
            p = p->next;
        }
    }

    if (!found) {
        return NULL;
    } else {
        match = p;
        for (p = head; p->next != match; p = p->next);
        p->next = match->next;
        match->next = NULL;
        return match;
    }
}
```