(i) (5 pts.)

Consider the following C program, which consists of two function definitions including the **`main`** function.

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

#define LENGTH (15)

void f(char *s, char *t, int i) {
    if (strlen(s) == 0) {
        t[i++] = '!';
        t[i] = '\0';
        return;
    } else if (isalpha(s[0])) {
        t[i++] = 'X';
    } else if (isdigit(s[0])) {
        t[i++] = 'Y';
    } else {
        t[i++] = 'Z';
    }
    f(&(s[1]), t, i);
}

int main(void) {
    char s1[LENGTH] = "h7v7???c";
    char s2[LENGTH] = "a99b88&&";

    f(s1, s2, 0);
    printf("s2 = [%s]\n", s2);
    f(s2, s1, 1);
    printf("s1 = [%s]\n", s1);

    return 0;
}
```

Show the complete output as it appears on standard output. <u>Show all work</u>, and clearly indicate your solution. Show your work and your solution for this problem <u>*only*</u> 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, a null pointer, or a pointer or array index 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.

This page is reserved as extra space for working on and writing your solution to Question (i).

(ii) (4 pts.)

Consider the following C program.

```
#include <stdio.h>

#define R 6
#define C 2
#define BASE (-1)

int M[R][C] = {-8, 2, -7, 4, -6, 6,
        7, 8, 9, 10, 11, 12};

int main(void) {
    int x = 0, y = 0, z = 0;

    for (x = 0; x < R; ++x) {
        z = BASE;
        for (y = 0; y < C; ++y) {
            if (M[x][y] < z) {
                z = M[x][y];
            }
        }
        printf("value #%d = %d\n", x + 1, z);
    }
    printf("\n");

    return 0;
}
```

Show the complete output as it appears on standard output. <u>Show all work</u>, and clearly indicate your solution. Show your work and your solution for this problem <u>*only*</u> 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, a null pointer, or a pointer or array index 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.

This page is reserved as extra space for working on and writing your solution to Question (ii).

(iii) (6 pts.)

Consider the following C program, which consists of a struct declaration, and three function definitions, including the `main` function.

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

struct elem {
    int val;
    struct elem *next;
};

struct elem **setup(int count) {
    struct elem **data = NULL;
    int i = 0, j = 0, val = 0;
    struct elem *item = NULL, *last = NULL;
    struct elem *link = NULL, *p = NULL;

    data = calloc(count, sizeof(struct elem *));
    for (i = 1; i < count; i++) {
        for (j = 1; j <= i; j++) {
            item = malloc(sizeof(struct elem));
            item->val = val;
            val++;
            item->next = NULL;
            if (p == NULL) {
                data[i] = item;
            } else {
                p->next = item;
            }
            p = item;
            if (((i % 2) == 1) && (j == 1)) {
                link = item;
            } else if (((i % 2) == 0) && (j == i)) {
                link = item;
            }
        }
        if (last == NULL) {
            data[i - 1] = link;
        } else {
            last->next = link;
        }
        last = item;
    }
    return data;
}
```

```
void traverse(struct elem *s, int n) {
    int i = 0;
    const int wrap = 6;

    for (i = 0; (i < n) && (s != NULL); i++) {
        if (i > 0) {
            if ((i % wrap) == 0) {
                printf("\n... ");
            } else {
                printf(", ");
            }
        }
        printf("%d", s->val);
        s = s->next;
    }
    printf("\n");
}

int main(void) {
    struct elem **e1 = NULL, **e2 = NULL;
    int s_count = 0, t_count = 0;

    s_count = 3;
    t_count = 3;
    e1 = setup(s_count);
    traverse(e1[0], t_count);
    printf("------\n");
    s_count = 9;
    t_count = 100;
    e2 = setup(s_count);
    traverse(e2[0], t_count);

    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 previous* page.

In case of an illegal dereferencing of a pointer (e.g., dereferencing of an uninitialized pointer, a null pointer, or a pointer or array index 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) (5 pts.)

Consider the following structure definition.

```
struct vect {
    int *elements;
    int length;
};
```

Consider also the following function prototype and its associated header comment.

```
/* Return the inner product of two vectors with circular traversal of the
vector with fewer elements (the shorter vector) in case the vectors do not
have equal lengths. If either vector length is less than or equal to zero,
then display an appropriate error message to standard error and exit the
program. If the vector lengths are equal, then return the inner product of
the two vectors. If the vector lengths are not equal, then traverse the
elements of the shorter vector in a circular fashion while computing the sum
of products of corresponding vector elements. For example,
if v1 = (3, 2, 4, 2, 5) and v2 = (8, 1), then the  function should return
3 x 8 + 2 x 1 + 4 x 8 + 2 x 1 + 5 x 8 = 100. */

int ip(struct vect *v1, struct vect *v2);
```

Develop a complete C code implementation of the function `ip`. You may define one or more additional functions that are called by `ip`.

Overflow checking in the sum of products computation is *not* required in your solution. You can also assume that `v1` and `v2` are non-NULL, so that NULL pointer checking is not required.

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

This page is reserved as extra space for working on and writing your solution to Question (iv).

Software Qualifying Exam Solutions

Spring 2019
Dept. of ECE, University of Maryland, College Park
11/20/2018

Problem 1:

```
s2 = [XYXYZZZX!]
s1 = [hXXXXXXXXZ!]
```

Problem 2:

```
value #1 = -8
value #2 = -7
value #3 = -6
value #4 = -1
value #5 = -1
value #6 = -1
```

Problem 3:

```
0, 2
------
0, 2, 3, 4, 5, 9
... 10, 11, 12, 13, 14, 20
... 21, 22, 23, 24, 25, 26
... 27, 35
```

Problem 4:

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

int ip_core(struct vect *small, struct vect *large) {
    int i = 0;
    int sum = 0;

    for (i = 0; i < large->length; i++) {
        sum += large->elements[i] *
                small->elements[i % (small->length)];
    }
    return sum;
}

int ip(struct vect *v1, struct vect *v2) {
    struct vect *small = NULL;
    struct vect *large = NULL;
    int result = 0;

    if ((v1->length <= 0) || (v2->length <= 0)) {
        fprintf(stderr, "Invalid vector length\n");
        exit(1);
    }

    if (v1->length < v2->length) {
        small = v1;
        large = v2;
    } else {
        small = v2;
        large = v1;
    }

    result = ip_core(small, large);
    return result;
}
```