(i) (5 pts.)

Consider the following C program, which includes two function definitions, including the main function.

```c
#include <stdio.h>

int f(int **w, int x, int *y, int z[2]) {
    x += 3;
    (*y) -= 7;
    z[0] += 2 + (**w)*2;
    return x + (*y) + z[0] - z[1];
}

int main(void) {
    int a = 3;
    int b = 2;
    int c[2] = {-5, 7};
    int *d = NULL;
    int r = 0;

    d = &c[1];
    r = f(&d, a, &b, c);
    printf("%d, %d, %d, %d, %d\n",
            a, b, c[0], c[1], r);
    d = &r;
    f(&d, b, &a, c);
    printf("%d, %d, %d, %d, %d\n",
            a, b, c[0], c[1], r);

    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.
This page is reserved as extra space for working on and writing your solution to Question (i)
(ii) (5 pts.)
Consider the following C program, which includes three function definitions, including the \textbf{main} function.

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

#define TOP 4
#define COUNT 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;
    t1 = malloc(sizeof(struct element));
    t2 = malloc(sizeof(struct element));
    t1->text = c1;
    t2->text = c2;
    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 = "Maryland", *name2 = "Virginia";
    int i = 0, wrap = 6;
    for (i = 1; i < TOP; i++) {
        data = add_elements(data, name1[i-1], name2[i+1]);
    }
    print_elements(data, 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 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.
(iii) (5 pts.)

Consider the following C program, which includes two function definitions, including the main function.

```c
#include <stdio.h>

struct vector {
    int length;
    int *values;
};

int vecop(struct vector *v1, struct vector *v2, int count) {
    int i = 0, sum = 0;
    if (count < 0) {
        return 1;
    } else if (count % 2 == 0) {
        return vecop(v1, v2, count - 1) + 2 * (v1->values)[count];
    } else {
        return vecop(v1, v2, count - 1) - (v2->values)[count];
    }
}

int main(void) {
    int a1[] = {4, 3, -1, 7, 0};
    int a2[] = {7, 2, 1};
    int a3[] = {1, 2, 0, 1, 2, 0};
    struct vector v1 = {5, a1};
    struct vector v2 = {3, a2};
    struct vector v3 = {6, a3};
    int result1 = 0, result2 = 0;
    result1 = vecop(&v1, &v2, 2);
    result2 = vecop(&v2, &v3, 2);
    printf("results: %d, %d\n", result1, result2);
    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.
This page is reserved as extra space for working on and writing your solution to Question (iii).
(iv) (5 pts.)

Write a function that takes a double value $x$, integer value $n$, and pointer $p$ as arguments, and sets the value pointed to by $p$ to be equal to the result of the following cosine power series computation.

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$$

The prototype of the function is as follows:

```c
void cosine_function(double x, double *p, int n);
```

Your solution should contain no function calls — i.e., no calls to any standard C library functions (including functions from math.h) or to any user-defined functions.

No error checking is required in the function.

Develop a complete C code implementation of the function `cosine_function`.

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.
This page is reserved as extra space for working on and writing your solution to Question (iv).
Software Qualifying Exam Solutions
Spring 2016
Dept. of ECE, University of Maryland, College Park
11/19/2015

Problem 1:

3, -5, 11, 7, 5
-4, -5, 23, 7, 5

Problem 2:

item 0: r
item 1: i
item 2: g
item 3: r
item 4: M
item 5: a
item 6: r
item 7: i

Problem 3:

results: 5, 15

Problem 4:

void cosine_function(double x, int n, double *p) {

double term = 1;
double factor = 1;
double factorial_update = 1;
double result = 1;
int i = 0;
double square = x * x;

for (i = 1; i <= n; i++) {
    factor *= (-1);
    factorial_update = (2 * i - 1) * (2 * i);
    term *= square / factorial_update;
    result += (term * factor);
}

(*p) = result;
}