SOFTWARE – Ph.D. Qualifying Exam Fall 2017

(i) (4 pts.)

Consider the following C program.

```
#include <stdio.h>

#define START 2
#define LIMIT 60
#define STEP 7
#define SIZE 3

int main(void) {
    int i = START, j = 0;
    int sum = 0, count = 0;
    int divisors[SIZE] = {4, 5, 7};

    while (i < LIMIT) {
        printf("%d\n", i);
        for (j = 0; j < SIZE; j++) {
            if ((i % divisors[j]) == 0) {
                sum += i;
                count++;
            }
        }
        i += STEP;
    }
    printf("count = %d, sum = %d\n", count, sum);

    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) (6 pts.)
Consider the following C program, which consists of three function definitions including the `main` function.

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

struct data {
    int v;
    struct data *next;
};

void setup(struct data **h, int *a, int len) {
    struct data *p = NULL;
    struct data *prev = NULL;
    struct data *start = NULL;
    int i = 0;
    for (i = 0; i < len; i++) {
        p = malloc(sizeof(struct data));
        p->v = a[i];
        if (prev == NULL) {
            start = p;
        } else {
            prev->next = p;
        }
        prev = p;
    }
    p->next = start;
    (*h) = start;
}

int f(struct data *h, int limit) {
    if (limit <= 0) {
        return h->v;
    } else {
        limit -= h->v;
        printf("[%d / %d]", h->v, limit);
        return h->v + f(h->next, limit);
    }
}

int main(void) {
    int values[] = {9, 3, 8, 3, 5, 5, 2};
    const int length1 = 7, length2 = 4, limit1 = 15, limit2 = 30;
    struct data *h = NULL;
    int result = 0;
    setup(&h, values, length1);
    result = f(h, limit1);
    printf("\nresult = %d\n", result);
    setup(&h, values, length2);
    result = f(h, limit2);
    printf("\nresult = %d\n", result);
    return 1;
}
```

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, 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.

```
#include <stdio.h>

int main(void) {
    char c1 = 'x';
    char *p1;
    char c2 = 'd';
    char *p2 = &c2;

    *p2 = c2 + 1;
    c1 = *p2 + 1;
    p1 = &c1;

    printf("c1 = %c, c2 = %c, *p1 = %c, *p2 = %c\n",
            c1, c2, *p1, *p2);

    return 0;
}
```

Suppose that each character occupies one byte of memory. Suppose also that the value assigned to `c1` is stored in (hexadecimal) address `D45` and the value assigned to `c2` is stored in address `CC7`.

PART A: Show the output of this program exactly 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.

PART B:
a) What value is represented by `&c1`?
b) What value is represented by `&c2`?
c) What value is assigned to `p1`?
d) What value is assigned to `p2`?

Show your work and your solution for this problem (both Parts A and B) *only* 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 (iii).

(iv) (5 pts.)

Consider the following function prototype and associated header comment.

```
/*********************************************************************
Perform a specified operation on a given string s, and display the
result on standard output. The name of the specified operation is
specified by the string pointed to by the "operation" argument. The
following are the valid (case-sensitive) operation names:
"test_numeric", "flip", and "capitalize". If the specified operation
name is not one of these valid names or if the string s is an empty
string (contains no non-null characters), then print a meaningful
error message to standard error. Otherwise, print to standard output
the result of applying the specified operation to the given string.
The result of the "test_numeric" operation is the string "true" if s
consists only of decimal numbers (that is, it contains only the
characters '0', '1', …, '9'); the result is the string "false"
otherwise. No modification is made to the string s in this case, only
the true/false result is printed to standard output. The result of the
flip operation is to reverse the characters in the string (for
example, "abcd" becomes ("dcba"). The result of the "capitalize"
operation is to convert all lower case alphabetic letters to their
corresponding upper case letters (for example 'c' becomes 'C');
characters that are not lower case letters are unchanged.
*********************************************************************/
void stringops(char *s, char *operation);
```

Note that no error checking is required in this function beyond what is stated in the header comment.

Part A: Consider the C program below, which illustrates correct usage of the function `stringops`.

```
int main(void) {
    char s1[] = "377966";
    char s2[] = "35xyx48";
    char s3[] = "A;;_bcD";

    stringops(s1, "test_numeric");
    stringops(s2, "test_numeric");
    stringops(s1, "flip");
    stringops(s2, "flip");
    stringops(s2, "capitalize");
    stringops(s3, "capitalize");
}
```

Show the complete output of this program as it appears on standard output.

Part B: Develop a complete C code implementation of the function **`stringops`**. In your implementation, make appropriate use of features that are provided through the standard library header file **`<string.h>`**.

Show your work and your solution for this problem (both Parts A and B) *only* on *this* page, the previous 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

Fall 2017
Dept. of ECE, University of Maryland, College Park
9/10/2017

Problem 1:

```
2
9
16
23
30
37
44
51
58
count = 3, sum = 90
```

Problem 2:

```
[9 / 6][3 / 3][8 / -5]
result = 23
[9 / 21][3 / 18][8 / 10][3 / 7][9 / -2]
result = 35
```

Problem 3:

PART A: c1 = f, c2 = e, *p1 = f, *p2 = e

PART B: (a) 0xD45, (b) 0xCC7, (c) 0xD45, (d) 0xCC7

Problem 4:

PART A:

```
true
false
669773
84xyx53
84XYX53
A;;_BCD
```

PART B:

```
void stringops(char *s, char *operation) {
    int length = strlen(s);
    int i = 0, j = 0;
    char tmp = '\0';

    if (length == 0) {
        fprintf(stderr, "empty string\n");
        return;
    }

    if (strcmp(operation, "test_numeric") == 0) {
        for (i = 0; i < length; i++) {
            if (!isdigit(s[i])) {
                printf("false\n");
                return;
            }
        }
        printf("true\n");
        return;
    } else if (strcmp(operation, "flip") == 0) {
        for (i = 0, j = (length -  1); i < j; i++, j--) {
            tmp = s[i];
            s[i] = s[j];
            s[j] = tmp;
        }
        printf("%s\n", s);
        return;
    } else if (strcmp(operation, "capitalize") == 0) {
        for (i = 0; i < length; i++) {
            s[i] = toupper(s[i]);
        }
        printf("%s\n", s);
        return;
    } else {
        fprintf(stderr, "invalid stringops operation\n");
    }
}
```