(i) (5 pts.)

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

```c
#include <stdio.h>

int f(int x) {
    if (x > 10) {
        return 2 + f(x - 4);
    } else if (x > 2) {
        return 1 + f(x - 3);
    } else {
        printf("x = %d\n", x);
        return x * x;
    }
}

int main(void) {
    int a = 14;
    int b = 0;

    b = f(a);

    printf("b = %d\n", b);

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

```c
#include <stdio.h>

void display_result(int result) {
    static int index = 1;
    printf("result #%d: %d\n", index, result);
    index++;
}

int main(void) {
    char *s = "xyzabc";
    char *t = "axyzapq";
    int iterations = 0, i = 0;

    iterations = 0;
    for (i = 0; i <= 17; i++) {
        iterations++;
    }
    display_result(iterations);

    iterations = 0;
    while (i > 0) {
        i -= 6;
        iterations++;
    }
    display_result(iterations);

    iterations = 0;
    for (i = 12; i > 4; i--) {
        i -= 3;
        iterations++;
    }
    display_result(iterations);

    iterations = 0;
    i = 0;
    do {
        iterations++;
        i++;
    } while (s[i - 1] == t[i]);
    display_result(iterations);

    iterations = 0;
    for (i = 8; (i / 3) != 0; i--) {
        iterations++;
    }
    display_result(iterations);

    return 0;
}
```
(iii) (5 pts.)

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

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

typedef struct elem {
    char c;
    struct elem *next;
    struct elem *prev;
} elem;

void insert_elem(elem **h, elem *e) {
    e->next = (*h);
    (*h)->prev = e;
    (*h) = e;
    e->prev = NULL;
}

elem *visit(elem *h, int forw, int back) {
    elem *p = h;
    int i = 0;
    for (i = 0; i < forw; i++) {
        p = p->next;
    }
    for (i = 0; i < back; i++) {
        p = p->prev;
    }
    return p;
}

int main(void) {
    elem a = {'A', NULL, NULL};
    elem *b = NULL; elem *head = &a;
    char s[] = "Maryland Terrapins";
    int size = 0;
    int i = 0;
    size = sizeof(elem);
    int length = 6;
    for (i = 0; i < length; i++) {
        b = malloc(size);
        b->c = s[(i + 14) % length];
        printf("inserting: %c\n", b->c);
        insert_elem(&head, b);
    }
    b = visit(head, 4, 3);
    printf("%c\n", b->c);
    b = visit(b, 3, 1);
    printf("%c\n", b->c);
    visit(b, 2, 3);
    printf("%c\n", b->c);
    return 0;
}
```
Consider the following function prototypes and their associated header comments.

```c
/* Return the total number of times that character c appears in string s. Return zero if c does not appear in string s. */
int string_count(char *s, char c);

/* Given that s and t are strings, set x[i] to be equal to the number of times that character t[i] appears in string s. Set x[i] in this way for i = 0, 1, ..., (L - 1), where L is the length of string t. Assume that storage for array x has been allocated before the function is called, and that array x has at least as many elements as there are characters in string t. */
void string_decompose(char *s, char *t, int x[]);
```

Part 1: Develop a complete C code implementation of the function `string_count`. Your implementation should NOT include calls to any functions or macros, not even functions or macros from the standard C library.

Part 2: Develop a complete C code implementation of the function `string_decompose` that makes use of the function `string_count` that you implemented in Part 1. Apart from calling `string_count`, your implementation should NOT include calls to any functions or macros, not even functions or macros from the standard C library.

Write your solution to this problem only on this page and (if more space is needed) the next page.
Problem 1:
\[ x = 1 \]
\[ b = 6 \]

Problem 2:

result #1: 18
result #2: 3
result #3: 2
result #4: 5
result #5: 6

Problem 3:

inserting: r
inserting: y
inserting: l
inserting: a
inserting: M
inserting: a
\(<M>\)
\(<l>\)
\(<l>\)
Problem 4:

```
#include <stdio.h>

int string_count(char *s, char c) {
    int count = 0;
    int i = 0;

    for (i = 0; s[i] != '\0'; i++) {
        if (s[i] == c) {
            count++;
        }
    }

    return count;
}

void string_decompose(char *s, char *t, int x[]) {
    int i = 0;

    for (i = 0; t[i] != '\0'; i++) {
        x[i] = string_count(s, t[i]);
    }
}
```
C Reference Card (ANSI)

**Program Structure/Functions**

```c
// C Reference Card (ANSI)

**Flow of Control**

```c

**C Preprocessor**

```c

**Data Types/Declarations**

```c

**Constants**

suffix: long, unsigned, float

65536L, -1U, 3.0F

**Flow Constructions**

```c

**ANSI Standard Libraries**

```c

**Character Class Tests**

```c

**String Operations**

```c

© 2007 Joseph H. Silverman Permissions on back. v2.2
Standard Utility Functions `<stdlib.h>`

- `abs(n)`
- `labs(n)`
- `div(n,d)`
- `ldiv(n,d)`
- `rand()`
- `srand(n)`
- `exit(status)`
- `vacall()`

Time and Date Functions `<time.h>`

- `ftime()`
- `ctime()`
- `tm_time` in seconds
- `difftime()`
- `mktime()`
- `asctime()`
- `strtime()`

Mathematical Functions `<math.h>`

- `sin(x)`
- `cos(x)`
- `tan(x)`
- `atan2(y,x)`
- `sqrt(x)`
- `exp(x)`
- `log(x)`
- `log10(x)`

Integer Type Limits `<limits.h>`

- `INT_MAX`
- `INT_MIN`
- `LONG_MAX`
- `LONG_MIN`
- `ULONGLONG_MAX`
- `ULLONG_MAX`

Float Type Limits `<float.h>`

- `FLT_RADIX`
- `FLT_ROUNDS`
- `FLT_EPSILON`
- `DBL_EPSILON`
- `DBL_DIG`
- `DBL_MINEXP`
- `DBL_MAXEXP`
- `DBL_MINEXP`