

SOFTWARE—Ph.D. Qualifying Exam Fall 2008

Question 1. [7 points] Write a function called `reverse_string` that takes a string as an argument, and reverses the characters of the string *in place*. Thus, the function changes the string that is passed to it so that the new string is the reversed version of the previous string. The prototype of the function is

```
void reverse_string(char *s);
```

Operation of the function is illustrated with the *main* function and corresponding output in Figure 1.

```
int main(void) {  
  
    char s1[] = "hello";  
    char s2[] = "abcd";  
    char s3[] = "";  
    char s4[] = "x";  
  
    reverse_string(s1);  
    reverse_string(s2);  
    reverse_string(s3);  
    reverse_string(s4);  
  
    printf("%s, %s, %s, %s\n", s1, s2, s3, s4);  
  
    return 0;  
}
```

Output:

```
olleh, dcba, , x
```

Fig. 1. A *main* function and its corresponding output to illustrate operation of the `reverse_string` function.

Write your solution to Question 1 on this page. Please clearly indicate your solution and show all work on this page.

Question 2. [9 points] Consider the following structure definition and associated documentation and pointer type definition.

```

/*****
A structure for a fraction. The numerator is a non-negative integer; the
denominator is a positive integer; and the sign is either 0 or 1. A sign of 1
indicates a negative-valued fraction, and a sign of 0 is used for zero- or
positive-valued fractions. A zero-valued fraction is any fraction that
has a zero-valued sign, a zero-valued numerator, and a positive-valued
denominator. It is an error for a 1-valued sign to be used with a zero-valued
numerator.

```

Note that fractions need not be reduced --- in other words, the numerator and denominator may contain common factors.

```

*****/
struct fraction {
    int numerator;
    int denominator;
    int sign;
};

typedef struct fraction *fraction_pointer;

```

Consider also the following function prototype and its associated documentation.

```

/*****
Given pointers to two fractions, add the two fractions; create a new fraction
that stores the sum; and return a pointer to this sum fraction. If the
denominators of fraction1 and fraction2 are equal, then the same denominator
should be used for the sum fraction; otherwise, the denominator of the sum
fraction should be the product of the denominators of fraction1 and fraction2.
*****/
fraction_pointer add_fractions(fraction_pointer fraction1,
                             fraction_pointer fraction2);

```

The operation of this function is illustrated by the *main* program and corresponding output in Figure 2.

Develop a complete the implementation of function `add_fractions`, and specify the `#include` directives that must be given at the top of the source code file.

```
int main(void) {  
  
    struct fraction f1;  
    struct fraction f2;  
    fraction_pointer result;  
  
    f1.numerator = 3;  
    f1.denominator = 5;  
    f1.sign = 1;  
  
    f2.numerator = 2;  
    f2.denominator = 4;  
    f2.sign = 0;  
  
    result = add_fractions(&f1, &f2);  
  
    printf("%d %d %d\n", result->sign, result->numerator,  
          result->denominator);  
  
    return 0;  
}
```

Output:

1 2 20

Fig. 2. A *main* function and its corresponding output to illustrate operation of the *add_fractions* function.

Write your solution to Question 2 on this page. Please clearly indicate your solution and show all work on this page.

Question 3. [4 points] Consider the following switch statement

```
switch (c) {
    case '+':
        result = x1 + x2;
        break;
    case '-':
        result = x1 - x2;
        break;
    case '*':
        result = x1 * x2;
        break;
    default:
        result = 0;
        break;
}
```

Write a new version of this code segment that has the same functionality but uses `if/else` statements instead of `switch` for the required program selection operations.

Write your solution to Question 3 on this page. Please clearly indicate your solution and show all work on this page.