Due Date: Tue., 6-dec-2011, 11:59PM

You may not share you code with anyone else, you must work alone! Feel free to discuss the project with anyone, just make sure you don't share code in any form! You are expected to write all the code you submit (do not submit code you find on the WWW!!!!).

Assignment

Write the C functions described below. It is essential that your C functions follow the names and arguments described below (we will write code that calls your functions!).

Your function definitions will be linked with our main code, so you just need to write the code for the two functions listed below. Feel free to write other functions as well, the only requirement is that you include the functions listed below. **DO NOT WRITE A main()!** It is provided below.

**Generating binary (ASCII encoded) from a char: print_binary_char**

```c
void print_binary_char(char x);
```

Write the function `print_binary_char`, which generates output to stdout that displays the binary representation of an 8 bit character `x`. For example, if `print_binary_char(0x35)` is called, the function should print the string `00110101`.

Your function must work with any `char` value `x`! The easiest way is to use binary logic operations to examine the bits of `x` one at a time. You might also find the shift operators useful.

**Generating binary (ASCII encoded) from an int: print_binary_int**

```c
void print_binary_int(int x);
```

Write the function `print_binary_int`, which generates output to stdout that displays the binary representation of a 32 bit integer `x`. For example, if `print_binary_int(0x01234567)` is called, the function should print the string: `00000001001000110100010101100111`.

Your function must work with any `int` value `x`! The easiest way to do this is to use your `print_binary_char` function on each of the bytes that make up the integer `x`. Be careful about byte ordering!

**Converting an ASCII encoded binary string to an int: get_int_from_binary**

```c
int get_int_from_binary(char *s);
```

Write the function `get_int_from_binary`, which looks at the string `s` (an ASCII string with exactly 32 characters, each of which is either '1' or '0') and converts it to a 32 bit C int (assume the data type int is 32 bits, and is signed).
Your function should **print** the value **and return** it!

For example, if your function was called like this:

```c
int x = get_int_from_binary("00000001001000110100010101100111");
```

it should return the value `0x01234567` (= 19088743 base 10).

If your function was called like this:

```c
int x = get_int_from_binary("11111111111111111111111111111111");
```

it should return the value `0xffffffff` (= -1).

### 32 bit signed addition with overflow detection: `safe_addition()`;

```c
int safe_addition(int x, int y, int *result);
```

This function adds two signed, 32 bit integers, and sets the memory location `result` to the sum. The function should return the value 1 if the result is correct, 0 if an overflow occurred. For example, adding `1000000000 + 2000000000` should result in an overflow.

### `main()`

We will be using the following `main` to test your functions, so you should not write a `main`. Use this main code to test each of your functions (by getting parameter value from the command line).

```c
#include <stdio.h>              /* printf, etc */
#include <stdlib.h>             /* atoi */
#include <string.h>             /* strcmp */

/* Function prototypes */

void print_binary_char(char x);
void print_binary_int(int x);
int get_int_from_binary(char *s);
int safe_addition(int x, int y, int *result);

int main(int argc, char **argv) {
    char c;
    int i;
    int x,y,z;

    if (argc<3) {
        printf("Invalid command line\n");
        printf("Usage: [-c decimalnum] | [-i decimalnum ] | [-b binarynum ]
|  [-a decimalnum decimalnum ]\n");
        exit(1);
    }
    if (strcmp(argv[1],"-c")==0) {
        // print byte value in binary
        c = atoi(argv[2]);
        printf("%d (0x%02x) : ",c,c);
```
print_binary_char(c);
printf("\n");
} else if (strcmp(argv[1],"-i")==0) {
  // print 32 bit int in binary
  i = atoi(argv[2]);
  printf("%d (0x%08x) : ",i,i);
  print_binary_int(i);
  printf("\n");
} else if (strcmp(argv[1],"-b")==0) {
  // convert from binary (ASCII encoded) to an int.
  i = get_int_from_binary(argv[2]);
  printf("%s: %d (%08x) \n",argv[2],i,i);
} else if ((argc >3) && (strcmp(argv[1],"-a")==0)) {
  // 32 bit addition with overflow check.
  x = atoi(argv[2]);
  y = atoi(argv[3]);
  if (safe_addition(x,y,&z)) {
    printf("%d + %d = %d \n",x,y,z);
  } else {
    printf("OVERFLOW: %d + %d \n",x,y);
  }
} else {
  printf("Invalid command line\n");
  printf("Usage: [-c decimalnum] | [-i decimalnum ] | [-b binarynum ]
| [-a decimalnum decimalnum ]\n");
  exit(1);
}
return(0);

Examples
Here are a few examples how to use the above main ('>' is the shell prompt).

> ./sampmain -c 100
100 (0x64) : 01100100
> ./sampmain -c -100
-100 (0xffffffff) : 10011100
> ./sampmain -i 1000
1000 (0x000003e8) : 0000000000000000000000111101000
> ./sampmain -i -1000
-1000 (0xffffffff) : 1111111111111111111111000011000
> ./sampmain -b 111111111111111111111111111111111111111111
10000000000000000000000000000000000000000000000000000
1000 (000003e8)
> ./sampmain -a 1000000000 2000000000
OVERFLOW: 1000000000 + 2000000000
> ./sampmain -a 2 3
2 + 3 = 5
How to submit

Use the submit command:

    submit cs203 s1.prog4 your_directory

Your programs should be under your_directory.
**Don't send compiled code, only send your C program!**

Grading

Grades will be determined by testing your functions with various parameter values. We will not test `get_int_from_binary` with anything other than a 32 character ASCII encoded binary strings (your code can assume that you are passing strings that contain the characters '0' and '1' only, and are the appropriate length).
Each function is worth 25% of the grade.

**HINTS:**

1. Review Chap. 2 of the textbook for integer/character representations and overflow/underflow.
2. The simplest approach is to manipulate bits (and groups of bits), rather than trying to use arithmetic.
3. Your code is to be run on a machine that uses a small-endian byte ordering. If you do something like this:

   ```c
   int i;
   char *s = (char *)&i;
   ```

   Remember that `s[0]` is not the MS byte in the `int`, (`s[3]` hold the MS byte).