Problem 1 *[MT1-1] Number Rep* (15 points)

Answer the following questions about number representation:

(a) Unsigned Base 4

(i) What is the range that a 4 digit unsigned base 4 number can represent? Write the bounds in decimal.

Solution: $0000_4 \sim 3333_4 = 0_{10} \sim 255_{10}$

(ii) Convert 107_{10} to unsigned base 4.

Solution: $107_{10} = 64 + 16 * 2 + 4 * 2 + 3 = 1223_4$

(b) Signed Base 4

(i) Suppose we wanted to use a bias in order to represent negative numbers in base4. If we are working with a 4 digit base 4 number, what should we choose as our bias? (Our bias should create equal amounts of negative and positive numbers for our range. If this is not possible, select a bias that will result in 1 more negative number than positive numbers). Express your answer in decimal.

Solution: 255/2 = 127. So the bias is -128 to favor negative numbers.

(ii) Suppose rather than using a bias notation, we decide to do the following.

For each base 4 number, we will reserve the most significant digit to strictly be used as a sign bit. A digit value of 1 will indicate a negative number, and a digit value of 0 will indicate a positive number. Any other values will result in an invalid number. For instance:

$$0003_4 = +3 \qquad 1003_4 = -3 \qquad 2003_4 = Invalid$$

How many valid representation can we represent with a 4 digit base 4 number using this scheme?

Solution: 2*4*4*4 = 128

(c) Given the following function in C:

```
int shifter(int x, int shift) {
    if (x > 0) {
        return x >> shift;
    }
    return -1 * (x >> shift);
}
```

Given y is a negative integer, and that `shifter(y, 2)` outputs 4, what is the range of values of y?

hint: `-8 >> 1 = -4`

Solution: -16 ~ -13

(d) Implement the function `unsigned int base_convert(unsigned int num, unsigned int base)`. This function takes in non-negative integers `num` and `base`. You are guaranteed the following:
- `base` is an integer in the range [2, 10], no need to error check this
- `num` is comprised of ”digits” with a value between 0 and `base` - 1.
- All values fit inside an `unsigned int`.

Your job is to make it so the function returns the decimal value of `num` in base `base`. For example, `base_convert(30, 4)` would return 12, since 30_4 is 12_{10}. You may not use additional lines (do not put multiple lines on the same line via ;) but you may not need all the lines provided. In addition, you may not include `<math.h>` or use `pow()`.

Solution:

```
unsigned int base_convert(unsigned int num, unsigned int base) {

    unsigned int value = 0, power = 1;

    while (num > 0) {
        value += (num % 10) * power;
        power *= base;
        num /= 10;
    }
    return value;
}
```

Problem 2 ***[MT1-2] Allocating an Order*** **(10 points)**

You are working on an e-commerce platform. Internally, orders are tracked through a struct called `order_t`. Your task is to write a function to allocate and initialize a new order. There's a catch though! This platform must be robust to errors, so you are required to return an error value from this function in addition to the newly allocated order. The possible errors are defined for you as preprocessor directives.

(a) **Write** `new_order`: Fill in the following code. Keep in mind the following requirements:

- You must return `BAD_ARG` if any inputs are invalid. The criteria for valid arguments is:
  - Unit price should be positive (no negative prices)
  - An order cannot be for more than `MAX_ORDER` items
  - Inputs must not cause your function to crash (or execute undefined behavior)
- You must return `NO_MEM` if there are any errors while allocating memory
- The tax rate is always initialized to `TAX_RATE`
- If your function returns `OK`, then `new` points to a valid struct that has been initialized with the provided values.

```
typedef struct order {
    int quantity;
    double unit_price;
    double tax_rate;
} order_t;

#define OK 0          /* Function executed correctly */
#define NO_MEM 1      /* Could not allocate memory for order */
#define BAD_ARG 2     /* An invalid argument was given */

#define TAX_RATE 1.08
#define MAX_ORDER 100
```

Solution:

```
/* Allocate and initialize a new order */
int new_order(order_t **new, int quantity, double unit_price) {
    /* Validate Arguments */
    if (new == NULL || quantity > MAX_ORDER || unit_price < 0)
        return BAD_ARG;

    /* Allocate "new" */
    *new = malloc(sizeof(order_t));
    if(*new == NULL) {
        return NO_MEM;

    /* Initialize "new" */
    (*new)->unit_price = unit_price;
    (*new)->quantity = quantity;
    (*new)->tax_rate = TAX_RATE;

    return OK;
}
```

(b) **Calling** `new_order`: How would you use `new_order()` to allocate and initialize `blue_monday` with a quantity of 10 and a unit price of 3.50 in the example below?

Solution:

```
order_t *blue_monday;
double total;
ret_t ret;

/* Fill in the arguments to new_order here */

ret = new_order(&blue_monday, 10, 3.50);

if (ret == OK) {
    printf("Total:  %lf\n",
            (blue_monday->unit_price *
                blue_monday->quantity * blue_monday->tax_rate));
} else {
    printf("Error\n");
}
```