Computer Systems Fundamentals

print_bits.h

```
// header file so we use print_bits in several examples
#ifndef PRINT_BITS_H

#include <stdint.h>
void print_bits(uint64_t value, int how_many_bits);

#endif
```

print_bits.c

two useful functions that we will use in a number of following programs

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

#include "print_bits.h"

// extract the nth bit from a value
int get_nth_bit(uint64_t value, int n) {
    // shift the bit right n bits
    // this leaves the n-th bit as the least significant bit
    uint64_t shifted_value = value >> n;

    // zero all bits except the the least significant bit
    int bit = shifted_value & 1;

    return bit;
}

// print the bottom how_many_bits bits of value
void print_bits(uint64_t value, int how_many_bits) {
    // print bits from most significant to least significant

    for (int i = how_many_bits - 1; i >= 0; i--) {
        int bit = get_nth_bit(value, i);
        printf("%d", bit);
    }
}
```

shift_as_multiply.c

Demonstrate that shifting the bits of a positive int left 1 position is equivalent to multiplying by 2

```
$ dcc shift_as_multiply.c print_bits.c -o shift_as_multiply
$ ./shift_as_multiply 4
2 to the power of 4 is 16

In binary it is: 00000000000000000000000000010000
$ ./shift_as_multiply 20
2 to the power of 20 is 1048576

In binary it is: 00000000000100000000000000000000
$ ./shift_as_multiply 31
2 to the power of 31 is 2147483648

In binary it is: 10000000000000000000000000000000
$
```

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "print_bits.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <exponent>\n", argv[0]);
        return 1;
    }

    int n = strtol(argv[1], NULL, 0);

    uint32_t power_of_two;

    int n_bits = 8 * sizeof power_of_two;

    if (n >= n_bits) {
        fprintf(stderr, "n is too large\n");
        return 1;
    }

    power_of_two = 1;
    power_of_two = power_of_two << n;

    printf("2 to the power of %d is %u\n", n, power_of_two);

    printf("In binary it is: ");
    print_bits(power_of_two, n_bits);
    printf("\n");

    return 0;
}
```

set_low_bits.c

Demonstrate use shift operators and subtraction to obtain a bit pattern of n 1s

```
$ dcc set_low_bits.c print_bits.c -o n_ones
$ ./set_low_bits 3

The bottom 3 bits of 7 are ones:
00000000000000000000000000000111
$ ./set_low_bits 19

The bottom 19 bits of 524287 are ones:
00000000000001111111111111111111
$ ./set_low_bits 29

The bottom 29 bits of 536870911 are ones:
00011111111111111111111111111111
```

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "assert.h"

#include "print_bits.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <exponent>\n", argv[0]);
        return 1;
    }

    int n = strtol(argv[1], NULL, 0);

    uint32_t mask;

    int n_bits = 8 * sizeof mask;

    assert(n >= 0 && n < n_bits);

    mask = 1;
    mask = mask << n;
    mask = mask - 1;

    printf("The bottom %d bits of %u are ones:\n", n, mask);
    print_bits(mask, n_bits);
    printf("\n");

    return 0;
}
```

set_bit_range.c

Demonstrate use shift operators and subtraction to obtain a bit pattern with a range of bits set.

```
$ dcc set_bit_range.c print_bits.c -o set_bit_range
$ ./set_bit_range 0 7

Bits 0 to 7 of 255 are ones:
00000000000000000000000011111111
$ ./set_bit_range 8 15

Bits 8 to 15 of 65280 are ones:
00000000000000001111111100000000
$ ./set_bit_range 8 23

Bits 8 to 23 of 16776960 are ones:
00000000111111111111111100000000
$ ./set_bit_range 1 30

Bits 1 to 30 of 2147483646 are ones:
01111111111111111111111111111110
```

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include "print_bits.h"

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <low-bit> <high-bit>\n", argv[0]);
        return 1;
    }

    int low_bit = strtol(argv[1], NULL, 0);
    int high_bit = strtol(argv[2], NULL, 0);

    uint32_t mask;

    int n_bits = 8 * sizeof mask;

    assert(low_bit >= 0);
    assert(high_bit >= low_bit);
    assert(high_bit < n_bits);

    int mask_size = high_bit - low_bit + 1;

    mask = 1;
    mask = mask << mask_size;
    mask = mask - 1;
    mask = mask << low_bit;

    printf("Bits %d to %d of %u are ones:\n", low_bit, high_bit, mask);
    print_bits(mask, n_bits);
    printf("\n");

    return 0;
}
```

extract_bit_range.c

Demonstrate use shift operators and subtraction to extract a bit pattern with a range of bits set.

```
$ dcc extract_bit_range.c print_bits.c -o extract_bit_range
$ ./extract_bit_range 4 7 42

Value 42 in binary is:
00000000000000000000000000101010

Bits 4 to 7 of 42 are:
0010
$ ./extract_bit_range 10 20 123456789

Value 123456789 in binary is:
00000111010110111100110100010101

Bits 10 to 20 of 123456789 are:
11011110011
```

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include "print_bits.h"

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <low-bit> <high-bit> <value>\n", argv[0]);
        return 1;
    }

    int low_bit = strtol(argv[1], NULL, 0);
    int high_bit = strtol(argv[2], NULL, 0);
    uint32_t value = strtol(argv[3], NULL, 0);

    uint32_t mask;

    int n_bits = 8 * sizeof mask;

    assert(low_bit >= 0);
    assert(high_bit >= low_bit);
    assert(high_bit < n_bits);

    int mask_size = high_bit - low_bit + 1;

    mask = 1;
    mask = mask << mask_size;
    mask = mask - 1;
    mask = mask << low_bit;

    // get a value with the bits outside the range low_bit..high_bit set to zero
    uint32_t extracted_bits = value & mask;

    // right shift the extracted bits so low_bit becomes bit 0
    extracted_bits = extracted_bits >> low_bit;

    printf("Value %u in binary is:\n", value);
    print_bits(value, n_bits);
    printf("\n");

    printf("Bits %d to %d of %u are:\n", low_bit, high_bit, value);
    print_bits(extracted_bits, mask_size);
    printf("\n");

    return 0;
}
```

bitwise.c

Demonstrate C bitwise operations of 16 bit values

```
$ dcc bitwise.c print_bits.c -o bitwise
$ ./bitwise

Enter a: 23032

Enter b: 12345

Enter c: 3
     a = 0101100111111000 = 0x59f8 = 23032
     b = 0011000000111001 = 0x3039 = 12345
    ~a = 1010011000000111 = 0xa607 = 42503
 a & b = 0001000000111000 = 0x1038 = 4152
 a | b = 0111100111111001 = 0x79f9 = 31225
 a ^ b = 0110100111000001 = 0x69c1 = 27073
a >> c = 0000101100111111 = 0x0b3f = 2879
a << c = 1100111111000000 = 0xcfc0 = 53184
```

```c
#include <stdio.h>
#include <stdint.h>
#include "print_bits.h"

void print_bits_hex(char *description, uint16_t n);

int main(void) {

    uint16_t a = 0;
    printf("Enter a: ");
    scanf("%hd", &a);

    uint16_t b = 0;
    printf("Enter b: ");
    scanf("%hd", &b);

    printf("Enter c: ");
    int c = 0;
    scanf("%d", &c);

    print_bits_hex("     a = ", a);
    print_bits_hex("     b = ", b);
    print_bits_hex("    ~a = ", ~a);
    print_bits_hex(" a & b = ", a & b);
    print_bits_hex(" a | b = ", a | b);
    print_bits_hex(" a ^ b = ", a ^ b);
    print_bits_hex("a >> c = ", a >> c);
    print_bits_hex("a << c = ", a << c);

    return 0;
}

// print description then print
// binary, hexadecimal and decimal representation of a value
void print_bits_hex(char *description, uint16_t value) {
    printf("%s", description);
    print_bits(value, 8 * sizeof value);
    printf(" = 0x%04x = %d\n", value & 0xFFFF, value);
}
```

print_int_in_hex.c

Print an integer in hexadecimal without using printf to demonstrate using bitwise operators to extract digits

```
$ dcc print_int_in_hex.c -o print_int_in_hex
$  ./print_int_in_hex

Enter a positive int: 42
42 = 0x0000002A
$  ./print_int_in_hex

Enter a positive int: 65535
65535 = 0x0000FFFF
$  ./print_int_in_hex

Enter a positive int: 3735928559
3735928559 = 0xDEADBEEF
$
```

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

void print_hex(uint32_t n);

int main(void) {

    uint32_t a = 0;
    printf("Enter a positive int: ");
    scanf("%u", &a);

    printf("%u = 0x", a);
    print_hex(a);
    printf("\n");

    return 0;
}

// print n in hexadecimal

void print_hex(uint32_t n) {

    // sizeof return number of bytes in n's representation
    // each byte is 2 hexadecimal digits

    int n_hex_digits = 2 * (sizeof n);

    // print hex digits from most significant to least significant

    for (int which_digit = n_hex_digits - 1; which_digit >= 0; which_digit--) {

        // shift value across so hex digit we want
        // is in bottom 4 bits

        int bit_shift = 4 * which_digit;
        uint32_t shifted_value = n >> bit_shift;

        // mask off (zero) all bits but the bottom 4 bites

        int hex_digit = shifted_value & 0xF;

        // hex digit will be a value 0..15
        // obtain the corresponding ASCII value
        // "0123456789ABCDEF" is a char array
        // containing the appropriate ASCII values (+ a '\0')

        int hex_digit_ascii = "0123456789ABCDEF"[hex_digit];

        putchar(hex_digit_ascii);
    }
}
```

[int_to_hex_string.c]

Convert an integer in hexadecimal to a string without using snprintf to demonstrate using bitwise operators to extract digits

```
$ dcc int_to_hex_string.c -o int_to_hex_string
$ ./int_to_hex_string
$ ./int_to_hex_string

Enter a positive int: 42
42 = 0x0000002A
$ ./int_to_hex_string

Enter a positive int: 65535
65535 = 0x0000FFFF
$ ./int_to_hex_string

Enter a positive int: 3735928559
3735928559 = 0xDEADBEEF
$
```

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

char *int_to_hex_string(uint32_t n);

int main(void) {
    uint32_t a = 0;
    printf("Enter a positive int: ");
    scanf("%u", &a);

    char *hex_string = int_to_hex_string(a);

    // print the returned string
    printf("%u = 0x%s\n", a, hex_string);

    free(hex_string);

    return 0;
}

// return a malloced string containing the hexadecimal digits of n

char *int_to_hex_string(uint32_t n) {
    // sizeof return number of bytes in n's representation
    // each byte is 2 hexadecimal digits

    int n_hex_digits = 2 * (sizeof n);

    // allocate memory to hold the hex digits + a terminating 0
    char *string = malloc(n_hex_digits + 1);

    // print hex digits from most significant to least significant

    for (int which_digit = 0; which_digit < n_hex_digits; which_digit++) {
        // shift value across so hex digit we want
        // is in bottom 4 bits

        int bit_shift = 4 * which_digit;
        uint32_t shifted_value = n >> bit_shift;

        // mask off (zero) all bits but the bottom 4 bites

        int hex_digit = shifted_value & 0xF;

        // hex digit will be a value 0..15
        // obtain the corresponding ASCII value
        // "0123456789ABCDEF" is a char array
        // containing the appropriate ASCII values

        int hex_digit_ascii = "0123456789ABCDEF"[hex_digit];

        string[which_digit] = hex_digit_ascii;
    }

    // 0 terminate the array
    string[n_hex_digits] = 0;

    return string;
}
```

hex_string_to_int.c

Convert a hexadecimal string to an integer

```
$ dcc hex_string_to_int.c -o hex_string_to_int
$ dcc hex_string_to_int.c -o hex_string_to_int
$ ./hex_string_to_int  2A
2A hexadecimal is 42 base 10
$ ./hex_string_to_int FFFF

FFFF hexadecimal is 65535 base 10
$ ./hex_string_to_int DEADBEEF

DEADBEEF hexadecimal is 3735928559 base 10
$
```

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

uint32_t hex_string_to_int(char *hex_string);
int hex_digit_to_int(int ascii_digit);

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <hexadecimal-number>\n", argv[0]);
        return 1;
    }

    char *hex_string = argv[1];

    uint32_t u = hex_string_to_int(hex_string);

    printf("%s hexadecimal is %u base 10\n", hex_string, u);

    return 0;
}

// print n in hexadecimal

uint32_t hex_string_to_int(char *hex_string) {
    uint32_t value = 0;

    for (int which_digit = 0; hex_string[which_digit] != 0; which_digit++) {
        int ascii_hex_digit = hex_string[which_digit];
        int digit_as_int = hex_digit_to_int(ascii_hex_digit);

        value = value << 4;
        value = value | digit_as_int;
    }

    return value;
}

// given the ascii value of a hexadecimal digit
// return the corresponding integer

int hex_digit_to_int(int ascii_digit) {
    if (ascii_digit >= '0' && ascii_digit <= '9') {
        // the ASCII characters '0' .. '9' are contiguous
        // in other words they have consecutive values
        // so subtract the ASCII value for '0' yields the corresponding integer

        return ascii_digit - '0';
    }

    if (ascii_digit >= 'A' && ascii_digit <= 'F') {
        // for characters 'A' .. 'F' obtain the
        // corresponding integer for a hexadecimal digit

        return 10 + (ascii_digit - 'A');
    }

    fprintf(stderr, "Bad digit '%c'\n", ascii_digit);
    exit(1);
}
```

pokemon.c

Represent a small set of possible values using bits

```c
#include <stdio.h>

#define FIRE_TYPE       0x0001
#define FIGHTING_TYPE   0x0002
#define WATER_TYPE      0x0004
#define FLYING_TYPE     0x0008
#define POISON_TYPE     0x0010
#define ELECTRIC_TYPE   0x0020
#define GROUND_TYPE     0x0040
#define PSYCHIC_TYPE    0x0080
#define ROCK_TYPE       0x0100
#define ICE_TYPE        0x0200
#define BUG_TYPE        0x0400
#define DRAGON_TYPE     0x0800
#define GHOST_TYPE      0x1000
#define DARK_TYPE       0x2000
#define STEEL_TYPE      0x4000
#define FAIRY_TYPE      0x8000

int main(void) {

    // give our pokemon 3 types
    int pokemon_type = BUG_TYPE | POISON_TYPE | FAIRY_TYPE;

    printf("0x%04xd\n", pokemon_type);

    if (pokemon_type & POISON_TYPE) {
        printf("Danger poisonous\n"); // prints
    }

    if (pokemon_type & GHOST_TYPE) {
        printf("Scary\n"); // does not print
    }

    return 0;
}
```

shift_bug.c

Examples of illegal bit-shift operations

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

int main(void) {
    // int16_t is a signed type (-32768..32767)
    // all operations below are defined for a signed type
    int16_t i;

    i = -1;
    i = i >> 1; // undefined - shift of a negative value
    printf("%d\n", i);

    i = -1;
    i = i << 1; // undefined - shift of a negative value
    printf("%d\n", i);

    i = 32767;
    i = i << 1; // undefined - left shift produces a negative value

    uint64_t j;
    j = 1 << 33; // undefined - constant 1 is an int
    j = ((uint64_t)1) << 33; // ok

    return 0;
}
```

xor.c

copy stdin to stdout xor'ing each byte with value given as argument

copy stdin to stdout xor'ing each byte with value given as argument

```
$ echo Hello Andrew|xor 42
bOFFE
kDNXO] $ echo Hello Andrew|xor 42|cat -A
bOFFE$
kDNXO] $
$  echo Hello |xor 42
bOFFE $ echo -n 'bOFFE '|xor 42

Hello
$ echo Hello|xor 123|xor 123

Hello
$
```

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <xor-value>\n", argv[0]);
        return 1;
    }

    int xor_value = strtol(argv[1], NULL, 0);

    if (xor_value < 0 || xor_value > 255) {
        fprintf(stderr, "Usage: %s <xor-value>\n", argv[0]);
        return 1;
    }

    int c;
    while ((c = getchar()) != EOF) {
        //    exclusive-or
        //    ^  | 0  1
        //    ---|-----
        //    0  | 0  1
        //    1  | 1  0

        int xor_c = c ^ xor_value;

        putchar(xor_c);
    }

    return 0;
}
```

bitset.c

Represent set of small non-negative integers using bit-operations

```
$ dcc bitset.c print_bits.c -o bitset
$ ./bitset

Set members can be 0-63, negative number to finish

Enter set a: 1 2 4 8 16 32 -1

Enter set b: 5 4 3 33 -1
a = 0000000000000000000000000000000100000000000000001000000100010110 = 0x100010116 = 4295033110
b = 0000000000000000000000000000000100000000000000000000000000111000 = 0x200000038 = 8589934648
a = {1,2,4,8,16,32}
b = {3,4,5,33}
a union b = {1,2,3,4,5,8,16,32,33}
a intersection b = {4}
cardinality(a) = 6
is_member(42, a) = 0
```

```c
#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include "print_bits.h"

typedef uint64_t set;

#define MAX_SET_MEMBER ((int)(8 * sizeof(set) - 1))
#define EMPTY_SET 0

set set_add(int x, set a);
set set_union(set a, set b);
set set_intersection(set a, set b);
set set_member(int x, set a);
int set_cardinality(set a);
set set_read(char *prompt);
void set_print(char *description, set a);

void print_bits_hex(char *description, set n);

int main(void) {
    printf("Set members can be 0-%d, negative number to finish\n",
           MAX_SET_MEMBER);
    set a = set_read("Enter set a: ");
    set b = set_read("Enter set b: ");

    print_bits_hex("a = ", a);
    print_bits_hex("b = ", b);
    set_print("a = ", a);
    set_print("b = ", b);
    set_print("a union b = ", set_union(a, b));
    set_print("a intersection b = ", set_intersection(a, b));
    printf("cardinality(a) = %d\n", set_cardinality(a));
    printf("is_member(42, a) = %d\n", (int)set_member(42, a));

    return 0;
}

set set_add(int x, set a) {
    return a | ((set)1 << x);
}

set set_union(set a, set b) {
    return a | b;
}

set set_intersection(set a, set b) {
    return a & b;
}

// return a non-zero value iff x is a member of a
set set_member(int x, set a) {
    assert(x >= 0 && x < MAX_SET_MEMBER);
    return a & ((set)1 << x);
}

// return size of set
int set_cardinality(set a) {
    int n_members = 0;
    while (a != 0) {
        n_members += a & 1;
        a >>= 1;
    }
    return n_members;
}

set set_read(char *prompt) {
    printf("%s", prompt);
    set a = EMPTY_SET;
    int x;
    while (scanf("%d", &x) == 1 && x >= 0) {
```

```
        a = set_add(x, a);
    }
    return a;
}

// print out member of the set in increasing order
// for example {5,11,56}
void set_print(char *description, set a) {
    printf("%s", description);
    printf("{");
    int n_printed = 0;
    for (int i = 0; i < MAX_SET_MEMBER; i++) {
        if (set_member(i, a)) {
            if (n_printed > 0) {
                printf(",");
            }
            printf("%d", i);
            n_printed++;
        }
    }
    printf("}\n");
}

// print description then binary, hex and decimal representation of value
void print_bits_hex(char *description, set value) {
    printf("%s", description);
    print_bits(value, 8 * sizeof value);
    printf(" = 0x%08lx = %ld\n", value, value);
}
```