# Computer Systems Fundamentals

digits.c

print digits from an integer one per line, reverse order

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int num;
    int rem;

    while (1) {  // forever
        // get the number
        printf("Integer? ");
        if (scanf("%d", &num) != 1) break;

        // extract the digits
        rem = num;
        do {
            printf("%d\n", rem % 10);
            rem = rem / 10;
        } while (rem != 0);
    }
    return 0;
}
```

bits.c

print bits from an integer one per line, reverse order

```c
#include <stdio.h>

#define MAXBITS 32

int main(int argc, char *argv[]) {
    int num;
    unsigned int rem;
    int bits[MAXBITS];
    int nbits;

    while (1) {  // forever
        // get the number
        printf("Integer? ");
        if (scanf("%d", &num) != 1) break;

        // extract the digits
        rem = num;
        nbits = 0;
        do {
            bits[nbits] = rem % 2;
            nbits++;
            rem = rem / 2;
        } while (rem != 0);

        printf("%d = %08x = ", num, num);
        for (int i = nbits-1; i >= 0; i--) {
            printf("%d", bits[i]);
        }
        putchar('\n');
    }
    return 0;
}
```

integer_prefixes.c

An example of inputing numbers in different bases and printing them out in different bases

```c
#include <stdio.h>

int main(void){

    unsigned int a = 938271;         //decimal
    unsigned int b = 0xAF12;         //hexadecimal
    unsigned int c = 0b00101010;     //binary (may not work on all systems)
    unsigned int d = 0123;           //0 octal

    printf("Dec: %u %u %u %u\n", a, b, c, d); //printing in decimal
    printf("Hex: %x %X %X %X\n", a, b, c, d); //printing in hex
    printf("Oct: %o %o %o %o\n", a, b, c, d); //printing in octal

    return 0;
}
```

integer_types.c

Print size and min and max values of integer types

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

int main(void) {

    char c;
    signed char sc;
    unsigned char uc;
    short s;
    unsigned short us;
    int i;
    unsigned int ui;
    long l;
    unsigned long ul;
    long long ll;
    unsigned long long ull;

    printf("%18s %5s %4s\n", "Type", "Bytes", "Bits");

    printf("%18s %5lu %4lu\n", "char",               sizeof c,   8 * sizeof c);

    printf("%18s %5lu %4lu\n", "signed char",        sizeof sc,  8 * sizeof sc);
    printf("%18s %5lu %4lu\n", "unsigned char",      sizeof uc,  8 * sizeof uc);

    printf("%18s %5lu %4lu\n", "short",              sizeof s,   8 * sizeof s);
    printf("%18s %5lu %4lu\n", "unsigned short",     sizeof us,  8 * sizeof us);

    printf("%18s %5lu %4lu\n", "int",                sizeof i,   8 * sizeof i);
    printf("%18s %5lu %4lu\n", "unsigned int",       sizeof ui,  8 * sizeof ui);

    printf("%18s %5lu %4lu\n", "long",               sizeof l,   8 * sizeof l);
    printf("%18s %5lu %4lu\n", "unsigned long",      sizeof ul,  8 * sizeof ul);

    printf("%18s %5lu %4lu\n", "long long",          sizeof ll,  8 * sizeof ll);
    printf("%18s %5lu %4lu\n", "unsigned long long", sizeof ull, 8 * sizeof ull);

    printf("\n");

    printf("%18s %20s %20s\n", "Type", "Min", "Max");

#ifdef __CHAR_UNSIGNED__
    printf("%18s %20hhu %20hhu\n", "char",               (char)CHAR_MIN,          (char)CHAR_MAX);
#else
    printf("%18s %20hhd %20hhd\n", "char",               (char)CHAR_MIN,          (char)CHAR_MAX);
#endif

    printf("%18s %20hhd %20hhd\n", "signed char",        (signed char)SCHAR_MIN,  (signed char)SCHAR_MAX);
    printf("%18s %20hhu %20hhu\n", "unsigned char",      (unsigned char)0,        (unsigned char)UCHAR_MAX);

    printf("%18s %20hd %20hd\n",   "short",              (short)SHRT_MIN,         (short)SHRT_MAX);
    printf("%18s %20hu %20hu\n",   "unsigned short",     (unsigned short)0,       (unsigned short)USHRT_MAX);

    printf("%18s %20d %20d\n",     "int",                INT_MIN,                 INT_MAX);
    printf("%18s %20u %20u\n",     "unsigned int",       (unsigned int)0,         UINT_MAX);

    printf("%18s %20ld %20ld\n",   "long",               LONG_MIN,                LONG_MAX);
    printf("%18s %20lu %20lu\n",   "unsigned long",      (unsigned long)0,        ULONG_MAX);

    printf("%18s %20lld %20lld\n", "long long",          LLONG_MIN,               LLONG_MAX);
    printf("%18s %20llu %20llu\n", "unsigned long long", (unsigned long long)0,   ULLONG_MAX);

    return 0;
}
```

overflow_int.c

This is an example of integer overflow
This is when we try to go beyond the limits of what a type can store
For ints this is undefined behaviour
On dcc you will get a run-time error with gcc it may wrap around and you will get incorrect numbers!

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

int main(void){
    int x = INT_MAX;
    printf("%d + 1 = ", x);
    x = x + 1;   //overflow undefined behaviour
    printf("%d\n", x);

    int y = INT_MIN;
    printf("%d - 1 = ", y);
    y = y - 1;   //overflow undefined behaviour
    printf("%d\n", y);

    return 0;
}
```

wrap_around_uint.c

This demonstrates the behaviour of unsigned ints that wrap around when we go beyond their limits
So adding 1 to the largest unsigned int, gives us 0
Subtracting 1 from 0 gives us the largest unsigned int

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

int main(void){
    unsigned int x = UINT_MAX;
    printf("%u + 1 = %u\n", x, x + 1);

    unsigned int y = -1;
    printf("-1 = %u\n", y);

    // infinite loop. Uncomment to try it out
    // z can never be < 0 as it is an unsigned type!
    // for(unsigned int z = 3; z >= 0; z--){
    //     printf("%u\n",z);
    // }

    return 0;
}
```

stdint.c

example declarations of the most commonly used fixed width integer types found in stdint.h

```c
#include <stdint.h>

int main(void) {

                    // range of values for type
                    //             minimum                 maximum
    int8_t   i1; //                  -128                     127
    uint8_t  i2; //                     0                     255
    int16_t  i3; //                -32768                   32767
    uint16_t i4; //                     0                   65535
    int32_t  i5; //           -2147483648              2147483647
    uint32_t i6; //                     0              4294967295
    int64_t  i7; // -9223372036854775808  9223372036854775807
    uint64_t i8; //                     0 18446744073709551615

    return 0;
}
```

char_bug.c

```c
#include <stdio.h>

int main(void) {

    // Common C bug:

    char c;  // c should be declared int   (int16_t would work, int is better)
    while ((c = getchar()) != EOF) {
        putchar(c);
    }

    // Typically `stdio.h` contains:
    // ```c
    // #define EOF -1
    // ```
    //
    // - most platforms: char is signed (-128..127)
    //   - loop will incorrectly exit for a byte containing 0xFF
    //
    // - rare platforms: char is unsigned (0..255)
    //   - loop will never exit

    return 0;
}
```

print_bits.h

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

```c
#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);
    }
}
```

print_bits_of_int.c

print the bits of an int, for example:

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

Enter an int: 42
00000000000000000000000000101010
$ ./print_bits_of_int

Enter an int: -42
11111111111111111111111111010110
$ ./print_bits_of_int

Enter an int: 0
00000000000000000000000000000000
$ ./print_bits_of_int

Enter an int: 1
00000000000000000000000000000001
$ ./print_bits_of_int

Enter an int: -1
11111111111111111111111111111111
$ ./print_bits_of_int

Enter an int: 2147483647
01111111111111111111111111111111
$ ./print_bits_of_int

Enter an int: -2147483648
10000000000000000000000000000000
$
```

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

int main(void) {
    int a = 0;
    printf("Enter an int: ");
    scanf("%d", &a);

    // sizeof returns number of bytes, a byte has 8 bits
    int n_bits = 8 * sizeof a;

    print_bits(a, n_bits);
    printf("\n");

    return 0;
}
```

8_bit_twos_complement.c

print the twos-complement representation of 8 bit signed integers essentially all modern machines represent integers in

```
$ dcc 8_bit_twos_complement.c print_bits.c -o 8_bit_twos_complement
$ ./8_bit_twos_complement
-128 10000000
-127 10000001
-126 10000010
-125 10000011
-124 10000100
-123 10000101
-122 10000110
-121 10000111
-120 10001000
-119 10001001
-118 10001010
-117 10001011
-116 10001100
-115 10001101
-114 10001110
-113 10001111
-112 10010000
-111 10010001
-110 10010010
-109 10010011
-108 10010100
-107 10010101
-106 10010110
-105 10010111
-104 10011000
-103 10011001
-102 10011010
-101 10011011
-100 10011100
 -99 10011101
 -98 10011110
 -97 10011111
 -96 10100000
 -95 10100001
 -94 10100010
 -93 10100011
 -92 10100100
 -91 10100101
 -90 10100110
 -89 10100111
 -88 10101000
 -87 10101001
 -86 10101010
 -85 10101011
 -84 10101100
 -83 10101101
 -82 10101110
 -81 10101111
 -80 10110000
 -79 10110001
 -78 10110010
 -77 10110011
 -76 10110100
 -75 10110101
 -74 10110110
 -73 10110111
 -72 10111000
 -71 10111001
 -70 10111010
 -69 10111011
 -68 10111100
 -67 10111101
 -66 10111110
 -65 10111111
 -64 11000000
 -63 11000001
 -62 11000010
 -61 11000011
```

```
 -60 11000100
 -59 11000101
 -58 11000110
 -57 11000111
 -56 11001000
 -55 11001001
 -54 11001010
 -53 11001011
 -52 11001100
 -51 11001101
 -50 11001110
 -49 11001111
 -48 11010000
 -47 11010001
 -46 11010010
 -45 11010011
 -44 11010100
 -43 11010101
 -42 11010110
 -41 11010111
 -40 11011000
 -39 11011001
 -38 11011010
 -37 11011011
 -36 11011100
 -35 11011101
 -34 11011110
 -33 11011111
 -32 11100000
 -31 11100001
 -30 11100010
 -29 11100011
 -28 11100100
 -27 11100101
 -26 11100110
 -25 11100111
 -24 11101000
 -23 11101001
 -22 11101010
 -21 11101011
 -20 11101100
 -19 11101101
 -18 11101110
 -17 11101111
 -16 11110000
 -15 11110001
 -14 11110010
 -13 11110011
 -12 11110100
 -11 11110101
 -10 11110110
  -9 11110111
  -8 11111000
  -7 11111001
  -6 11111010
  -5 11111011
  -4 11111100
  -3 11111101
  -2 11111110
  -1 11111111
   0 00000000
   1 00000001
   2 00000010
   3 00000011
   4 00000100
   5 00000101
   6 00000110
   7 00000111
   8 00001000
   9 00001001
  10 00001010
  11 00001011
```

```
 12 00001100
 13 00001101
 14 00001110
 15 00001111
 16 00010000
 17 00010001
 18 00010010
 19 00010011
 20 00010100
 21 00010101
 22 00010110
 23 00010111
 24 00011000
 25 00011001
 26 00011010
 27 00011011
 28 00011100
 29 00011101
 30 00011110
 31 00011111
 32 00100000
 33 00100001
 34 00100010
 35 00100011
 36 00100100
 37 00100101
 38 00100110
 39 00100111
 40 00101000
 41 00101001
 42 00101010
 43 00101011
 44 00101100
 45 00101101
 46 00101110
 47 00101111
 48 00110000
 49 00110001
 50 00110010
 51 00110011
 52 00110100
 53 00110101
 54 00110110
 55 00110111
 56 00111000
 57 00111001
 58 00111010
 59 00111011
 60 00111100
 61 00111101
 62 00111110
 63 00111111
 64 01000000
 65 01000001
 66 01000010
 67 01000011
 68 01000100
 69 01000101
 70 01000110
 71 01000111
 72 01001000
 73 01001001
 74 01001010
 75 01001011
 76 01001100
 77 01001101
 78 01001110
 79 01001111
 80 01010000
 81 01010001
 82 01010010
 83 01010011
```

```
  84 01010100
  85 01010101
  86 01010110
  87 01010111
  88 01011000
  89 01011001
  90 01011010
  91 01011011
  92 01011100
  93 01011101
  94 01011110
  95 01011111
  96 01100000
  97 01100001
  98 01100010
  99 01100011
 100 01100100
 101 01100101
 102 01100110
 103 01100111
 104 01101000
 105 01101001
 106 01101010
 107 01101011
 108 01101100
 109 01101101
 110 01101110
 111 01101111
 112 01110000
 113 01110001
 114 01110010
 115 01110011
 116 01110100
 117 01110101
 118 01110110
 119 01110111
 120 01111000
 121 01111001
 122 01111010
 123 01111011
 124 01111100
 125 01111101
 126 01111110
 127 01111111
$
```

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

int main(void) {

    for (int i = -128; i < 128; i++) {
        printf("%4d ", i);
        print_bits(i, 8);
        printf("\n");
    }

    return 0;
}
```

endian.c

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

int main(void) {
    uint8_t b;
    uint32_t u;

    u = 0x03040506;
    // load first byte of u
    b = *(uint8_t *)&u;
    // prints 6 if little-endian
    // and 3 if big-endian
    printf("%d\n", b);
}
```

endian.s

```
main:
    lbu  $a0, u       # b = *(uint8_t *)&u;
    li   $v0, 1       # printf("%d", a0);
    syscall

    li   $a0, '\n'    # printf("%c", '\n');
    li   $v0, 11
    syscall

    li   $v0, 0       # return 0
    jr   $ra

    .data
u:
    .word 0x3040506  #u = 0x03040506;
```