Information Processing 2 — Week 4 exercises

Complete the exercises in-class on your own laptop. You do not need to submit the answers online. Instead, ask an instructor to check your work during the class as soon as you finish answering each question. If you do not finish all the questions in-class, try to finish them outside class and have them checked *at the start of the next exercises class*.

1 [1 point] Printing binary numbers

The following function `printNumber(int n)` prints the number `n` in decimal.

```
void printNumber(int i)
{
    if (i >= 10) printNumber(i / 10);
    putchar('0' + i % 10);
}
```

Modify the function to print `n` in binary (base 2) instead of decimal. Test your function like this:

```
int main()
{
    int i;
    for (i = 0;  i <= 20;  ++i) {
        printNumber(i);
        printf("\n");
    }
    return 0;
}
```

Verify that the output is: `0 1 10 11 100 ... 10001 10010 10011 10100`

2 [2 points] Printing digits in (almost) any base

Write a function `void printDigit(int n)` that prints the number `n` as a single digit. If `d < 10` then it should print a single digit between '0' and '9'. If `d >= 10` then should print a single letter starting from '`A`' (representing the digit with value ten). Test your function like this:

```
int main()
{
    int i;
    for (i = 0;  i <= 20;  ++i) {
        printDigit(i);
        printf("\n");
    }
    return 0;
}
```

Verify that the output is: `0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K`

3 [1 point] Printing hexadecimal numbers

Modify `printNumber()` so that it prints the number `n` in hexadecimal. Hint: your `printDigit()` function will be useful. Test `printNumber()` by printing numbers from 0 to 20. Verify that the output is: `0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14`

4 [1 point] Printing numbers in any base

Write a function `void printNumber(int n, int b)` that prints the number `n` in base `b`. Test your function by printing the number 42 in every base from 2 to 20. Verify that the output is:

```
101010 1120 222 132 110 60 52 46 42 39 36 33 30 2C 2A 28 26 24 22
```

5 [1 point] Printing signed numbers

Modify your `printNumber()` function so that it prints negative numbers properly. Test it by printing −42 in base 16, which is '−2A'.

6 [1 point] Reading digit characters as digit values

Write a function `int digitValue(int c)` that returns the value of the decimal digit represented by character `c`. For all other characters it should return −1 to indicate 'not a digit'. Verify that the digit values for characters between `'/'` and `':'` (inclusive) are: `-1 0 1 2 3 4 5 6 7 8 9 -1`

7 [2 points] Reading digit values in (almost) any base

Add a second parameter to `int digitValue(int c, int b)` so that it returns a value between 0 and `b`−1 for a character `c` that is valid digit in base `b`. Allow letters to represent digits whose value is larger than 9 (so `'A'` or `'a'` is digit value 10, `'B'` or `'b'` is digit value 11, and so on). Your function should continue to return −1 if the character `c` is not a valid digit in base `b`.

Test your function by reading hexadecimal digits from the input and printing the decimal value of each digit. Verify that both upper-case (capital) and lower-case (small) letters work as hexadecimal digits. Verify that non-hexadecimal digits have value −1.

8 [1 point] Converting strings to numbers

Use `digitValue(int c, int b)` to write a function `strtoi(char s[], int b)` that converts a base `b` number in the string `s` into its value. For example, `strtoi("2a", 16)` should return 42. Verify that:

```
strtoi("1010",  2) ==   10
strtoi("1010",  8) ==  520
strtoi("1010", 10) == 1010
strtoi("1010", 16) == 4112
strtoi("2A",   16) ==   42
```

Your `strtoi` function should stop when it encounters an invalid digit. Verify that:

```
strtoi("1012",  2) ==   5
strtoi("1789",  8) ==  15
strtoi("101A", 10) == 101
strtoi("2FGH", 16) ==  47
```

9 Challenge: [1 bonus point] Reading signed numbers

Extend `strtoi()` so that it ignores leading blanks (spaces or tabs) and then skips any number of '+' and '−' sign characters before it processes the digits. The value returned should have the correct sign (positive or negative) according to the number of '−' characters that were read before the first digit. Test your program with the following `main` program and verify that each result is correct. (If you cannot figure out why a particular result is correct, convert the hexadecimal digits to binary and then calculate the value 'by hand' as a **signed** 32-bit number.)

```
int main()
{
    printf("%d\n", strtoi("  101010",    2));
    printf("%d\n", strtoi(" +52",        8));
    printf("%d\n", strtoi("++42",       10));
    printf("%d\n", strtoi("  2a",       16));
    printf("%d\n", strtoi(" -2A",       16));
    printf("%d\n", strtoi("--2A",       16));
    printf("%d\n", strtoi("  7fffffff", 16));
    printf("%d\n", strtoi("  ffffffff", 16));
    printf("%d\n", strtoi(" -ffffffff", 16));
    printf("%d\n", strtoi("  80000000", 16));
    printf("%d\n", strtoi(" -80000000", 16));

    return 0;
}
```