Information Processing 2 — Week 6 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*.

Note: Exercises checked **more than one week** after the class will have a **50% late penalty** applied.

Note: for each exercise you can download a 'template' from Teams that contains the test program shown in this document.

Multi-way branches using `switch`

In previous weeks you have written programs to count different things using a short sequence of conditional '`if`' statements. When there are more than a few different things to count, the sequence of '`if`' statements becomes long and slow. A '`switch`' statement can often do the same thing, faster and much more neatly.

1 Counting many different kinds of characters

Write a program that counts digits, letters, newlines, tabs, spaces, and 'other' characters. Your program should have the following features:

- Six counter variables `d`, `l`, `n`, `t`, `s`, and `o` for counting digits, letters, newlines, tabs, spaces, and other characters.
- An 'infinite' `for (;;) { ... }` loop to process characters from the input one at a time.
- A call to `getchar()` at the start of the loop body to read the next character, followed by...
- A conditional statement that tests for `EOF` and uses `break` to leave the `for` loop at end-of-file.
- A `switch()` on the character that was read, and several `case`s to determine whether it was
  - a digit `'0'...'9'`
  - a letter `'A'...'Z'` or `'a'...'z'`
  - a newline `'\n'`
  - a tab `'\t'`
  - a space `' '`
  - something other than the above (the `default` case)

  (with each `case` of the `switch` incrementing the corresponding counter variable).

Hints: To detect and count the 'other' characters you can use a '`default:`' case within the `switch`. Don't forget the `break` at the end of every `case`.

Test your function using the following program (download '`06-01-counting.c`' from Teams):

```
int main()
{
    int d = 0, l = 0, n = 0, t = 0, s = 0, o = 0;
    // your loop and switch statement go here :)
    printf("digits:   %3d\n", d);
    printf("letters:  %3d\n", l);
    printf("tabs:     %3d\n", t);
    printf("newlines: %3d\n", n);
    printf("spaces:   %3d\n", s);
    printf("others:   %3d\n", o);

    return 0;
}
```

Download '06-test.txt' from Teams and use it to test your program. Verify that the output is:

```
$ ./06-01-counting < 06-test.txt
digits:    16
letters:  658
tabs:       2
newlines:  12
spaces:   121
others:    22
```

▷▷ Ask an instructor to check and record your work.

Converting integers to strings

Write a function that converts an `int` into a string in a general way, providing many of the same same options as the '%d' conversion specifier of `printf()`:

```
int itosnbwp(int i, char s[], int n, int b, int w, int p)
```

- `i` is the integer to print,
- `s` is a character array that will hold the string representing `i`,
- `n` is the maximum number of characters in `s`,
- `b` is the base to be used to represent `i`,
- `w` is the field width in which `i` will be printed, right-justified, and
- `p` is the character used to 'pad' the field when `i` has fewer than `w` digits.

This is a lot to write all at once, so the next 8 exercises take you through the development one step at a time.

2 Converting digit values to digit characters

Write a function `int digitChar(int d)` that converts the digit value `d` into a character. The function should return a character in the range `'0'` through `'9'` for the first ten digit values, and a character in the range `'A'` through `'Z'` for digit values 10 through 35. (If the digit value is outside these ranges, return the character `'?'`.)

Test your function using the following program (download '06-02-digitchar.c' from Teams):

```
int main(int argc, char *argv[])
{
    for (int i = 0;  i < 40;  ++i)
        putchar(digitChar(i));
    putchar('\n');

    return 0;
}
```

Verify that the output is: `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ????`

▷▷ Ask an instructor to check and record your work.

3 Printing the digits of a number using a loop

Write a function `void iprint(int i)` that uses `digitChar()` to print each digit in the integer `i` in reverse order (i.e., backwards). Hints:

- Do not use a recursive function. Use a loop instead.
- Each time around the loop, print the least significant digit (the 'units' column) then shift `i` right one digit (using division).
- Since you should print at least one digit, even when `i` is initially `0`, choose a kind of loop that always runs the body at least once.

Test your function using the following program (download '`06-03-iprint.c`' from Teams):

```
int main(int argc, char *argv[])
{
    iprint(    0);
    iprint(    1);
    iprint(   42);
    iprint(65535);

    return 0;
}
```

Verify that the output is: `0 1 24 53556`

▷▷ Ask an instructor to check and record your work.

4 Storing the digits of a number in a string

Modify `iprint()` to make the function `int itor(int i, char s[])` that converts `i` into a sequence of digits in reverse order and stores them in `s` (instead of printing them). The function should return the number of digits stored in the string. (Don't forget to nul-terminate the string.)

Test your function using the following program (download '`06-04-itor.c`' from Teams):

```
int main(int argc, char *argv[])
{
    char s[1024];

    itor(    0, s);  printf("%s\n", s);
```

```
        itor(    1, s);  printf("%s\n", s);
        itor(   42, s);  printf("%s\n", s);
        itor(65535, s);  printf("%s\n", s);

        return 0;
    }
```

Verify that the output is still: `0 1 24 53556`

▷▷ Ask an instructor to check and record your work.

5 Reversing the characters in a string

Write a function `void reverse(char s[], int len)` that reverses the first `len` characters in the string `s`. Hint: Swap the first and last, then the second and second last, and so on. Stop when you reach the middle of the string.

Modify `itor()` from the previous exercise to make the function `int itos(int i, char s[])` that converts `i` into a sequence of digits in the correct order. Hint: Construct the string in the wrong order, just like you did with `itor()`, then use `reverse()` at the end of the function to put the characters into the correct order.

Test your function using the following program (download '`06-05-itos.c`' from Teams):

```
    int main(int argc, char *argv[])
    {
        char s[1024];

        itos(    0, s);  printf("%s\n", s);
        itos(    1, s);  printf("%s\n", s);
        itos(   42, s);  printf("%s\n", s);
        itos(65535, s);  printf("%s\n", s);

        return 0;
    }
```

Verify that the output is now: `0 1 42 65535`

▷▷ Ask an instructor to check and record your work.

6 Preventing buffer over-run

Your `itos()` function is not safe. If the string `s` is too small to hold the result then `itos()` will write characters beyond the end of `s`, with results that could crash your program.

Modify `itos()` to make the function `int itosn(int i, char s[], int n)` which converts `i` into a string, storing at most `n` characters in `s` (including the terminating nul character). Hint: You can test `n` at the start of the loop and `break` out of it if `n` is not at least `1` (to guarantee space to store the terminating nul character).

Test your function using the following program (download '`06-06-itosn.c`' from Teams):

```
int main(int argc, char *argv[])
{
    char s[1024];

    itosn(    0, s, 4);  printf("%s\n", s);
    itosn(    1, s, 4);  printf("%s\n", s);
    itosn(   42, s, 4);  printf("%s\n", s);
    itosn(65535, s, 4);  printf("%s\n", s);

    return 0;
}
```

Verify that the output is: `0 1 42 535`

▷▷ Ask an instructor to check and record your work.

7 Converting integers to strings in any base

Add another parameter to make the function `itosnb(int i, char s[], int n, int b)` that uses base `b` instead of base 10.

Test your function using the following program (download '`06-07-itosnb.c`' from Teams):

```
int main(int argc, char *argv[])
{
    char s[1024];

    itosnb(    0, s, 1024,  2);  printf("%s\n", s);
    itosnb(    1, s, 1024,  2);  printf("%s\n", s);
    itosnb(   42, s, 1024,  2);  printf("%s\n", s);
    itosnb(   42, s, 1024,  8);  printf("%s\n", s);
    itosnb(   42, s, 1024, 10);  printf("%s\n", s);
    itosnb(65535, s, 1024, 16);  printf("%s\n", s);

    return 0;
}
```

Verify the output is:

```
0
1
101010
52
42
FFFF
```

▷▷ Ask an instructor to check and record your work.

8 Converting integers to strings within a specified field width

Add another parameter to make `itosnbw(int i, char s[], int n, int b, int w)` that converts the integer `i` into a string right-justified in a field width of `w` characters, padded on the left with spaces.

Hint: When you have stored all the digits (in reverse order), continue storing spaces into the string until it is w characters long or there is only space for one more character (the terminating nul). Then store the nul and reverse the contents of the string. The additional spaces will now be on the left, with the number right-justified.

Test your function using the following program (download '06-08-itosnbw.c' from Teams):

```
int main(int argc, char *argv[])
{
    char s[1024];

    itosnbw(    0, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(    1, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024,  8, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024, 10, 4);  printf("%s\n", s);
    itosnbw(65535, s, 1024, 16, 4);  printf("%s\n", s);

    return 0;
}
```

Verify the output is:

```
   0
   1
101010
  52
  42
FFFF
```

▷▷ Ask an instructor to check and record your work.

9 Converting integers to strings with a specified padding character

Add a parameter to make itosnbwp(int i, char s[], int n, int b, int w, int p) so that it converts the integer i into a string in s in base b, right-justified in a field width of w characters, padded on the left with character p.

Test your function using the following program (download '06-09-itosnbwp.c' from Teams):

```
int main(int argc, char *argv[])
{
    char s[1024];

    itosnbwp(    0, s, 1024,  2, 4, ' ');  printf("%s\n", s);
    itosnbwp(    1, s, 1024,  2, 4, '0');  printf("%s\n", s);
    itosnbwp(   42, s, 1024,  2, 4, ' ');  printf("%s\n", s);
    itosnbwp(   42, s, 1024,  8, 4, ' ');  printf("%s\n", s);
    itosnbwp(   42, s, 1024, 10, 4, '0');  printf("%s\n", s);
    itosnbwp(65535, s, 1024, 16, 4, '0');  printf("%s\n", s);

    return 0;
}
```

Verify the output is:

```
   0
0001
101010
  52
0042
FFFF
```

▷▷ Ask an instructor to check and record your work.

Writing specific functions using more general functions

The C library contains many variations of the `printf()` function, including `fprintf()` (write the output to a file), `sprintf()` (write the output to a string), `snprintf()` (write the output to a string with a specified maximum length), and so on. Most of these functions are implemented using one very general function that can be specialised (using specific choices of argument values) to perform each of the 'simpler' functions.

10 Implementing most of your functions using one general function

You now have a very general function `itosnbwp` which can be used to implement all of the previous functions. Implement

- `itosnbw` using `itosnbwp` (by passing an appropriate final argument to it),
- `itosnb` using `itosnbw` (by passing an appropriate final argument to it),
- `itosn` using `itosnb` (by passing an appropriate final argument to it),
- `itos` using `itosn` (by passing an appropriate final argument to it), and
- `iprint` using `itos` (printing the digits in the *correct* order).

Test your functions using the following program (download '`06-10-itosall.c`' from Teams):

```c
int main(int argc, char *argv[])
{
    char s[1024];

    iprint(    0);
    iprint(    1);
    iprint(   42);
    iprint(65535);

    itos(    0, s);  printf("%s\n", s);
    itos(    1, s);  printf("%s\n", s);
    itos(   42, s);  printf("%s\n", s);
    itos(65535, s);  printf("%s\n", s);

    itosn(    0, s, 4);  printf("%s\n", s);
    itosn(    1, s, 4);  printf("%s\n", s);
    itosn(   42, s, 4);  printf("%s\n", s);
    itosn(65535, s, 4);  printf("%s\n", s);

    itosnb(    0, s, 1024,  2);  printf("%s\n", s);
    itosnb(    1, s, 1024,  2);  printf("%s\n", s);
    itosnb(   42, s, 1024,  2);  printf("%s\n", s);
    itosnb(   42, s, 1024,  8);  printf("%s\n", s);
    itosnb(   42, s, 1024, 10);  printf("%s\n", s);
    itosnb(65535, s, 1024, 16);  printf("%s\n", s);

    itosnbw(    0, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(    1, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024,  2, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024,  8, 4);  printf("%s\n", s);
    itosnbw(   42, s, 1024, 10, 4);  printf("%s\n", s);
    itosnbw(65535, s, 1024, 16, 4);  printf("%s\n", s);

    itosnbwp(    0, s, 1024,  2, 4, ' ');  printf("%s\n", s);
    itosnbwp(    1, s, 1024,  2, 4, '0');  printf("%s\n", s);
    itosnbwp(   42, s, 1024,  2, 4, ' ');  printf("%s\n", s);
    itosnbwp(   42, s, 1024,  8, 4, ' ');  printf("%s\n", s);
    itosnbwp(   42, s, 1024, 10, 4, '0');  printf("%s\n", s);
    itosnbwp(65535, s, 1024, 16, 4, '0');  printf("%s\n", s);

    return 0;
}
```

Verify the output is:

```
0
1
42
65535
0
1
42
65535
0
1
42
535
0
1
101010
52
42
FFFF
   0
   1
101010
  52
  42
FFFF
   0
0001
101010
  52
0042
```

```
FFFF
```

▷▷ Ask an instructor to check and record your work.

11 Bonus: Implementing loops with goto and labels

The `do`, `while`, and `for` loops can easily be implemented with `goto` and labels. The following example shows how to implement a `do` loop (including a `continue` statement in its body) using only labels and `goto` statements.

```
// do loop

i = 0;
do {
    if (i == 3) continue;
    printf("%2d", i);
} while (i++ < 5);
printf("\n");
```

```
  // do loop using labels and goto

  i = 0;
doBody:
  if (i == 3) goto doTest;
  printf("%2d", i);
doTest:
  if (i++ < 5) goto doBody;
  printf("\n");
```

Download '`06-11-loops.c`' from Teams. It contains the above `do` loop example, plus a `while` loop and a `for` loop. Complete the program by adding implementations of the `while` loop and `for` loop using only labels and `goto`. Verify the output is:

```
$ ./06-11-loops
 0 1 2 4 5
 0 1 2 4 5
 1 2 4 5
 1 2 4 5
 0 1 2 4
 0 1 2 4
```

▷▷ Ask an instructor to check and record your work.