Chapter 51: <ctype.h> — character classification & conversion

Section 51.1: Introduction

The header `ctype.h` is a part of the standard C library. It provides functions for classifying and converting characters.

All of these functions take one parameter, an `int` that must be either *EOF* or representable as an unsigned char.

The names of the classifying functions are prefixed with 'is'. Each returns an integer non-zero value (TRUE) if the character passed to it satisfies the related condition. If the condition is not satisfied then the function returns a zero value (FALSE).

These classifying functions operate as shown, assuming the default C locale:

```
int a;
int c = 'A';
a = isalpha(c); /* Checks if c is alphabetic (A-Z, a-z), returns non-zero here. */
a = isalnum(c); /* Checks if c  is alphanumeric (A-Z, a-z, 0-9), returns non-zero here. */
a = iscntrl(c); /* Checks is c is a control character (0x00-0x1F, 0x7F), returns zero here. */
a = isdigit(c); /* Checks if c is a digit (0-9), returns zero here. */
a = isgraph(c); /* Checks if c has a graphical representation (any printing character except space),
returns non-zero here. */
a = islower(c); /* Checks if c is a lower-case letter (a-z), returns zero here. */
a = isprint(c); /* Checks if c is any printable character (including space), returns non-zero here.
*/
a = isupper(c); /* Checks if c is a upper-case letter (a-z), returns zero here. */
a = ispunct(c); /* Checks if c is a punctuation character, returns zero here. */
a = isspace(c); /* Checks if c is a white-space character, returns zero here. */
a = isupper(c); /* Checks if c is an upper-case letter (A-Z), returns non-zero here. */
a = isxdigit(c); /* Checks if c is a hexadecimal digit (A-F, a-f, 0-9), returns non-zero here. */
```

Version ≥ C99

```
a = isblank(c); /* Checks if c is a blank character (space or tab), returns non-zero here. */
```

There are two conversion functions. These are named using the prefix 'to'. These functions take the same argument as those above. However the return value is not a simple zero or non-zero but the passed argument changed in some manner.

These conversion functions operate as shown, assuming the default C locale:

```
int a;
int c = 'A';

/* Converts c to a lower-case letter (a-z).
 * If conversion is not possible the unchanged value is returned.
 * Returns 'a' here.
 */
a = tolower(c);

/* Converts c to an upper-case letter (A-Z).
 * If conversion is not possible the unchanged value is returned.
 * Returns 'A' here.
 */
a = toupper(c);
```

The below information is quoted from cplusplus.com mapping how the original 127-character ASCII set is considered by each of the classifying type functions (a • indicates that the function returns non-zero for that character)

ASCII values	characters	iscntrl	isblank	isspace	isupper	islower	isalpha	isdigit	isxdigit	isalnum	ispunct	isgraph	isprint
0x00 .. 0x08	NUL, (other control codes)	•											
0x09	tab ('\t')	•	•	•									
0x0A .. 0x0D	(white-space control codes: '\f','\v','\n','\r')	•		•									
0x0E .. 0x1F	(other control codes)	•											
0x20	space (' ')		•	•									•
0x21 .. 0x2F	!"#$%&'()*+,-./										•	•	•
0x30 .. 0x39	0123456789							•	•	•		•	•
0x3a .. 0x40	:;<=>?@										•	•	•
0x41 .. 0x46	ABCDEF				•		•		•	•		•	•
0x47 .. 0x5A	GHIJKLMNOPQRSTUVWXYZ				•		•			•		•	•
0x5B .. 0x60	[]^_`										•	•	•
0x61 .. 0x66	abcdef					•	•		•	•		•	•
0x67 .. 0x7A	ghijklmnopqrstuvwxyz					•	•			•		•	•
0x7B .. 0x7E	{}~bar										•	•	•
0x7F	(DEL)	•											

Section 51.2: Classifying characters read from a stream

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

typedef struct {
  size_t space;
  size_t alnum;
  size_t punct;
} chartypes;

chartypes classify(FILE *f) {
  chartypes types = { 0, 0, 0 };
  int ch;

  while ((ch = fgetc(f)) != EOF) {
    types.space += !!isspace(ch);
    types.alnum += !!isalnum(ch);
    types.punct += !!ispunct(ch);
  }

  return types;
}
```

The `classify` function reads characters from a stream and counts the number of spaces, alphanumeric and punctuation characters. It avoids several pitfalls.

- When reading a character from a stream, the result is saved as an `int`, since otherwise there would be an ambiguity between reading `EOF` (the end-of-file marker) and a character that has the same bit pattern.
- The character classification functions (e.g. `isspace`) expect their argument to be *either representable as an* `unsigned char`*, or the value of the* `EOF` *macro*. Since this is exactly what the `fgetc` returns, there is no need for conversion here.
- The return value of the character classification functions only distinguishes between zero (meaning **`false`**) and nonzero (meaning **`true`**). For counting the number of occurrences, this value needs to be converted to a 1 or 0, which is done by the double negation, `!!`.

Section 51.3: Classifying characters from a string

```
#include <ctype.h>
#include <stddef.h>

typedef struct {
  size_t space;
  size_t alnum;
  size_t punct;
} chartypes;

chartypes classify(const char *s) {
  chartypes types = { 0, 0, 0 };
  const char *p;
  for (p= s; p != '\0'; p++) {
    types.space += !!isspace((unsigned char)*p);
    types.alnum += !!isalnum((unsigned char)*p);
    types.punct += !!ispunct((unsigned char)*p);
  }

  return types;
}
```

The `classify` function examines all characters from a string and counts the number of spaces, alphanumeric and punctuation characters. It avoids several pitfalls.

- The character classification functions (e.g. `isspace`) expect their argument to be *either representable as an* `unsigned char`*, or the value of the* `EOF` *macro*.
- The expression `*p` is of type `char` and must therefore be converted to match the above wording.
- The `char` type is defined to be equivalent to either `signed char` or `unsigned char`.
- When `char` is equivalent to `unsigned char`, there is no problem, since every possible value of the `char` type is representable as `unsigned char`.
- When `char` is equivalent to `signed char`, it must be converted to `unsigned char` before being passed to the character classification functions. And although the value of the character may change because of this conversion, this is exactly what these functions expect.
- The return value of the character classification functions only distinguishes between zero (meaning **`false`**) and nonzero (meaning **`true`**). For counting the number of occurrences, this value needs to be converted to a 1 or 0, which is done by the double negation, `!!`.