Programmation C et Système
Les fonctions de `<ctype.h>`

Régis Barbanchon

L2 Informatique

Les fonctions de `<ctype.h>`

L'entête déclare plusieurs fonctions de test sur les caractères :

- `ispunct()` : teste si un caractère est un signe de ponctuation.
- `isspace()` : teste si un caractère est une espace.
- `isdigit()` : teste si un caractère est un chiffre décimal.
- `isxdigit()` : teste si un caractère est un chiffre hexadécimal.
- `islower()` : teste si un caractère est une lettre minuscule.
- `isupper()` : teste si un caractère est une lettre majuscule.
- `isalpha()` : teste si un caractère est alphabétique.
- `isalnum()` : teste si un caractère est alphanumérique.
- `isgraph()` : teste si un caractère est graphique.
- `isprint()` : teste si un caractère est d'affichage.
- `iscntrl()` : teste si un caractère est de contrôle.

Il déclare aussi deux fonctions de conversion de caractère :

- `tolower()` : retourne le caractère en minuscule.
- `toupper()` : retourne le caractère en majuscule.

Les prototypes de <ctype.h>

Les prototypes des fonctions de test sont tous identiques :

```
int ispunct (int character);
int isspace (int character);
int isdigit (int character);
int isxdigit(int character);
int islower (int character);
int isupper (int character);
```

```
int isalpha (int character);
int isalnum (int character);
int isgraph (int character);
int isprint (int character);
int iscntrl (int character);
```

- Le `int` retourné est une valeur Booléenne non-normalisée (la valeur *vrai* est non-nulle mais pas nécessairement `1`).
- `character` est de type `int` car il peut valoir `EOF` (*end of file*). S'il ne vaut pas `EOF`, il doit contenir un `unsigned char`.
- Tous les tests `isXXXXX(EOF)` retournent `0` (la valeur *faux*).

Les prototypes des fonctions de conversions semblent similaires :

```
int tolower(int character);
```

```
int toupper(int character);
```

mais le `int` retourné contient soit `EOF` soit un `unsigned char`, avec `tolower(EOF)` et `toupper(EOF)` retournant `EOF`.

Les prédicats `isspace()` et `ispunct()`

Dans les locales `"C"` et `"POSIX"`, le prédicat `isspace()` accepte :

- l'espace (code ASCII 32, `' '`).
- les tabulations horizontale (9, `'\t'`) et verticale (11, `'\v'`),
- le *Carriage-Return* (13, `'\r'`), et le *Line-Feed* (10, `'\n'`),
- le *Form-Feed* (12, `'\f'`),

et le prédicat `ispunct()` accepte les caractères affichables qui ne sont ni des lettres, ni des chiffres, ni le caractère espace.

```
void CTypeTest_isspace_ispunct (void)
{
  char * oldCtype= setlocale (LC_CTYPE, NULL);
  setlocale (LC_CTYPE, "C");

  char spaces[]= "\t\v\r\n\f ";
  char puncts[]= "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~";
  for (int c= 1; c < 128; c++) {
    assert (!! isspace (c)  == !! strchr (spaces, c));
    assert (!! ispunct (c)  == !! strchr (puncts, c));
  }
  assert ( ! isspace (EOF)); assert ( ! isspace ('\0'));
  assert ( ! ispunct (EOF)); assert ( ! ispunct ('\0'));

  setlocale (LC_CTYPE, oldCtype);
}
```

Les prédicats `isdigit()` et `isxdigit()`

Dans les locales `"C"` et `"POSIX"`, le prédicat `isdigit()` accepte :

- les chiffres de `'0'` à `'9'`,

et le prédicat `isxdigit()` accepte en plus de ces chiffres :

- les 6 lettres minuscules de `'a'` à `'f'`,
- les 6 lettres majuscules de `'A'` à `'F'`.

```
void CTypeTest_isdigit_isxdigit (void)
{
  char * oldCtype= setlocale (LC_CTYPE, NULL);
  setlocale (LC_CTYPE, "C");

  char digits []= "0123456789";
  char xdigits[]= "0123456789ABCDEFabcdef";
  for (int c= 1; c < 128; c++) {
    assert (!! isdigit (c)  == !! strchr (digits,  c));
    assert (!! isxdigit(c)  == !! strchr (xdigits, c));
  }
  assert ( ! isdigit (EOF)); assert ( ! isdigit ('\0'));
  assert ( ! isxdigit(EOF)); assert ( ! isxdigit('\0'));

  setlocale (LC_CTYPE, oldCtype);
}
```

Les prédicats `islower()` et `isupper()`

Dans les locales `"C"` et `"POSIX"`, le prédicat `islower()` accepte :

- les 26 lettres minuscules de `'a'` à `'z'`,

et le prédicat `isupper()` accepte :

- les 26 lettres majuscules de `'A'` à `'Z'`.

```
void CTypeTest_islower_isupper (void)
{
  char * oldCtype= setlocale (LC_CTYPE, NULL);
  setlocale (LC_CTYPE, "C");

  char lowers[]= "abcdefghijklmnopqrstuvwxyz";
  char uppers[]= "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  for (int c= 1; c < 128; c++) {
    assert (!! islower (c)  == !! strchr (lowers, c));
    assert (!! isupper (c)  == !! strchr (uppers, c));
  }
  assert ( ! islower (EOF)); assert ( ! islower ('\0'));
  assert ( ! isupper (EOF)); assert ( ! isupper ('\0'));

  setlocale (LC_CTYPE, oldCtype);
}
```

Les autres prédicats `isxxx()` dérivés

Les autres prédicats sont dérivés : un caractère `c` est accepté par :

- `isalpha(c)` ssi `islower(c)` ou `isupper(c)` l'acceptent,
- `isalnum(c)` ssi `isalpha(c)` ou `isdigit(c)` l'acceptent,
- `isgraph(c)` ssi `isalnum(c)` ou `ispunct(c)` l'acceptent,
- `isprint(c)` ssi `isgraph(c)` l'accepte ou `c == ' '`,
- `iscntrl(c)` ssi `isprint(c)` le rejette.

```
void CTypeTest_isxxx (void)
{
  char * oldCtype= setlocale (LC_CTYPE, NULL);
  setlocale (LC_CTYPE, "C");
  for (int c= 0; c < 128; c++) {
    assert (!! isalpha (c) == (isupper (c) || islower (c)));
    assert (!! isalnum (c) == (isalpha (c) || isdigit (c)));
    assert (!! isgraph (c) == (isalnum (c) || ispunct (c)));
    assert (!! isprint (c) == (isgraph (c) || c == ' '));
    assert (!! iscntrl (c) == ! isprint(c));
  }
  setlocale (LC_CTYPE, oldCtype);
}
```

Remarque : le code ci-dessus l'idiome de la double négation (`!!`) pour normaliser les valeurs Booléennes retournées par `isxxx()`.

Les fonctions de conversion `tolower()` et `toupper()`

Enfin, il y a deux fonctions de conversion sur les caractères :

- `tolower(c)` retourne la version minuscule de `c`.
- `toupper(c)` retourne la version majuscule de `c`.

```
void CTypeTest_tolower_toupper (void) {
  char * oldCtype= setlocale (LC_CTYPE, NULL);
  setlocale (LC_CTYPE, "C");
  char lowers[]= "abcdefghijklmnopqrstuvwxyz";
  char uppers[]= "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  for (int c= 1; c < 128; c++) {
    char * lowerOcc= strchr (lowers, c);
    char * upperOcc= strchr (uppers, c);
    if (lowerOcc != NULL) {
      int index= lowerOcc - lowers;
      assert (toupper(c) == uppers [index]);
      assert (tolower(c) == c);
    } else if (upperOcc != NULL) {
      int index= upperOcc - uppers;
      assert (tolower(c) == lowers [index]);
      assert (toupper(c) == c);
    } else {
      assert (tolower(c) == c);
      assert (toupper(c) == c);
    }
  }
  assert (tolower (EOF) == EOF); assert (tolower ('\0') == '\0');
  assert (toupper (EOF) == EOF); assert (toupper ('\0') == '\0');
  setlocale (LC_CTYPE, oldCtype);
}
```