UNIVERSITÉ PARIS-XII — FACULTÉ DE SCIENCES ÉCONOMIQUES ET DE GESTION
Master Professionnel *Méthodes Appliquées de la Statistique et de l'Économétrie pour la Recherche, l'Analyse et le Traitement de l'Information*
Année universitaire 2006-2007 — *François LEGENDRE*
F.Legendre@univ-paris12.fr — http://fj.legendre.free.fr

Projet « Répertoire »

1 Exposé du projet

Il est assez fréquent de vouloir parcourir l'une après l'autre les différentes entrées d'un répertoire. Par exemple, on veut supprimer tous les fichiers temporaires (dont l'extension serait par exemple ***tmp***) qui figurent dans un répertoire. Pour cela, il faut itérer sur toutes les entrées de ce répertoire ; si l'entrée est un fichier et si l'extension de son nom est la chaîne de caractères ***"tmp"*** alors il faut détruire le fichier.

On veut reproduire l'interface de lecture des lignes dans un fichier. Le programmeur est en effet habitué à programmer, pour itérer sur les lignes d'un fichier, comme suit.

```
#include <fstream>
ifstream fichier("nom de mon fichier") ;
if ( ! fichier ) { // Oups, il n'est pas possible d'ouvrir le fichier.
  exit(1) ; }
string ligne ;
while ( getline(fichier, ligne) ) {
  // Traitement qui utilise 'ligne'.
}
```

On veut donc pouvoir écrire le fragment de programme suivant.

```
#include <directory>
Directory directory("nom de mon répertoire") ;
if ( ! directory ) { // Oups, il n'est pas possible de lire les entrées du répertoire.
  exit(1) ; }
string nom ;
while ( getentry(directory, nom) ) {
  // Traitement qui utilise 'nom'.
}
```

Cette interface est contestable ; elle n'est pas vraiment « orientée objet ». Elle est pourtant simple et efficace.

Le but de ce projet est de montrer de quelle façon il est possible d'encapsuler des fonctions de la bibliothèque standard du langage C qui permettent, justement, de parcourir les entrées d'un répertoire. C'est habituellement au système d'exploitation de rendre ce genre de services. Le système UNIX a rendu populaire un certain nombre de fonctions que l'on retrouve sur presque tous les systèmes d'exploitation. Il s'agit des fonctions ***opendir(...)***, ***readdir(...)*** et ***closedir(...)***.

À titre d'exemple, le programme ci-après illustre l'utilisation de notre bibliothèque. Le programme décompte le nombre de fichiers d'extension ***tmp*** dans le répertoire « ***C:/DocTeX***» et dans ses sous-répertoires.

```
#include <iostream>   // 'cin', 'cout' et 'cerr'.
#include <string>     // Chaînes de caractères de la STL.
#include "directory.h" // Bibliothèque de parcours des répertoires.

using namespace std ;

size_t nombre_fichiers = 0 ;

void
visiter_et_compter(const string & chemin)
{
  cout << "Visite du répertoire : " << chemin << endl ;
  Directory directory(chemin) ;
  if ( ! directory ) {
    cerr << "Impossible d'ouvrir en lecture le répertoire '" << chemin << "'." << endl ;
    exit(1) ; }
  string nom ;
  while ( getentry(directory, nom) ) {
    if ( (nom == ".") || (nom == "..") ) {
      continue ; }
    if ( isdirectory(chemin + '/' + nom) ) {
      visiter_et_compter(chemin + '/' + nom) ; } // Appel récursif.
    else {
      // Ne pas dépasser le début de chaîne !
      string extension(nom.end()-min(size_t(4), nom.size()), nom.end()) ;
      if ( (extension == ".tmp") || (extension == ".TMP") ) {
        ++ nombre_fichiers ; } } }
}
int
main()
{
  visiter_et_compter("c:/DocTeX") ;
  cout << "Nombre de fichiers d'extension 'tmp'\t" << nombre_fichiers << endl ;

}
```

Ci-après figure la documentation de la fonction ***readdir(...)*** telle qu'on peut la trouver sur Internet.

NAME ***readdir, readdir_r - read directory***

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);

DESCRIPTION The type ***DIR***, which is defined in the header ***<dirent.h>***, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files ; files may be removed from a directory or added to a directory asynchronously to the operation of ***readdir()***.

The ***readdir()*** function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument ***dirp***, and positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. The structure ***dirent*** defined by the ***<dirent.h>*** header describes a directory entry.

If entries for dot or dot-dot exist, one entry will be returned for dot and one entry will be returned for dot-dot ; otherwise they will not be returned.

The pointer returned by ***readdir()*** points to data which may be overwritten by another call to ***readdir()*** on the same directory stream. This data is not overwritten by another call to ***readdir()*** on a different directory stream.

If a file is removed from or added to the directory after the most recent call to ***opendir()*** or ***rewinddir()***, whether a subsequent call to ***readdir()*** returns an entry for that file is unspecified.

The ***readdir()*** function may buffer several directory entries per actual read operation ; ***readdir()*** marks for update the ***st_atime*** field of the directory each time the directory is actually read.

After a call to ***fork()***, either the parent or child (but not both) may continue processing the directory stream using ***readdir()***, ***rewinddir()*** or ***seekdir()***. If both the parent and child processes use these functions, the result is undefined.

If the entry names a symbolic link, the value of the ***d_ino*** member is unspecified.

The ***readdir()*** interface need not be reentrant.

The ***readdir_r()*** function initialises the ***dirent*** structure referenced by ***entry*** to represent the directory entry at the current position in the directory stream referred to by ***dirp***, store a pointer to this structure at the location referenced by ***result***, and positions the directory stream at the next entry.

The storage pointed to by ***entry*** will be large enough for a ***dirent*** with an array of ***char d_name*** member containing at least ***{NAME_MAX}*** plus one elements.

On successful return, the pointer returned at ****result*** will have the same value as the argument ***entry***. Upon reaching the end of the directory stream, this pointer will have the value ***NULL***.

The ***readdir_r()*** function will not return directory entries containing empty names. It is unspecified whether entries are returned for dot or dot-dot.

If a file is removed from or added to the directory after the most recent call to ***opendir()*** or ***rewinddir()***, whether a subsequent call to ***readdir_r()*** returns an entry for that file is unspecified.

The ***readdir_r()*** function may buffer several directory entries per actual read operation ; the ***readdir_r()*** function marks for update the ***st_atime*** field of the directory each time the directory is actually read.

Applications wishing to check for error situations should set ***errno*** to 0 before calling ***readdir()***. If ***errno*** is set to non-zero on return, an error occurred.

RETURN VALUE Upon successful completion, ***readdir()*** returns a pointer to an object of type ***struct dirent***. When an error is encountered, a null pointer is returned and ***errno*** is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and ***errno*** is not changed.

If successful, the ***readdir_r()*** function returns zero. Otherwise, an error number is returned to indicate the error.

ERRORS

1. The ***readdir()*** function will fail if :
   (a) ***[EOVERFLOW]*** One of the values in the structure to be returned cannot be represented correctly.
2. The ***readdir()*** function may fail if :
   (a) ***[EBADF]*** The ***dirp*** argument does not refer to an open directory stream.
   (b) ***[ENOENT]*** The current position of the directory stream is invalid.
3. The ***readdir_r()*** function may fail if :
   (a) ***[EBADF]*** The ***dirp*** argument does not refer to an open directory stream.

EXAMPLES The following sample code will search the current directory for the entry ***name*** :

```
dirp = opendir(".");

while (dirp) {
  errno = 0;
  if ((dp = readdir(dirp)) != NULL) {
```

```
      if (strcmp(dp->d_name, name) == 0) {
        closedir(dirp);
        return FOUND;
      }
    } else {
      if (errno == 0) {
        closedir(dirp);
        return NOT_FOUND;
      }
      closedir(dirp);
      return READ_ERROR;
    }
  }

  return OPEN_ERROR;
```

APPLICATION USAGE The ***readdir()*** function should be used in conjunction with ***opendir()***, ***closedir()*** and ***rewinddir()*** to examine the contents of the directory.

FUTURE DIRECTIONS None.

SEE ALSO ***closedir()***, ***lstat()***, ***opendir()***, ***rewinddir()***, ***symlink()***, ***<dirent.h>***, ***<sys/types.h>***.

DERIVATION ***readdir_r()*** derived from the POSIX Threads Extension (1003.1c-1995).

Cette documentation n'est pas particulièrement claire et il vous appartient d'être suffisamment coriace pour pouvoir pleinement utiliser ce genre de documentation.

Il apparaît toutefois qu'il est nécessaire d'appeler tout d'abord la fonction ***opendir(...)*** ensuite la fonction ***readdir(...)*** et enfin la fonction ***closedir(...)***. La fonction ***opendir(...)*** retourne un pointeur sur une structure de type ***DIR***; ce pointeur est ensuite utilisé comme argument des fonctions ***readdir(...)*** et ***closedir(...)***.

Notre objet ***Directory*** va donc encapsuler ce fameux pointeur, dans sa section ***private***. La fonction ***opendir(...)*** sera appelée dans le constructeur; la fonction ***closedir(...)*** dans le destructeur. L'interface prescrit que la lecture de l'entrée suivante s'obtient par l'appel d'une fonction et non d'une méthode. On va utiliser la syntaxe « ***getentry(directory, nom)*** » et non la syntaxe « ***directory.getentry(nom)*** ». La fonction ***getentry(..., ...)*** doit cependant pouvoir accéder aux détails de l'implémentation. Elle est donc déclarée ***friend*** dans la définition de l'objet ***Directory***. Aussi la fonction ***getentry(..., ...)*** va-t-elle disposer des mêmes droits qu'une méthode qui aurait été déclarée dans la définition de ***Directory***.

Le fichier ***directory.h*** est destiné à être inclu par la directive « ***#include "directory.h"*** ». À ce stade, le programmeur n'a sans doute pas utilisé une instruction ***using***. Il faut donc complètement qualifier les identificateurs qui interviennent dans le fichier. C'est ainsi que l'argument du constructeur est déclaré comme un « ***const std::string &*** ».

Le traitement des erreurs est assuré en utilisant l'expression « ***! directory*** ». Le programmeur code les lignes suivantes.

```
if ( ! directory ) {
  ... traitement de l'erreur ...
}
```

Le fragment de programme suivant est erroné.

```
if ( directory ) {
  ... traitement quant tout va bien ...
} else {
  ... traitement de l'erreur ...
}
```

En effet, l'expression « ***! directory*** » a un sens parce que l'opérateur ***operator!*** a été défini pour une instance de l'objet ***Directory***. En revanche, l'expression « ***directory*** » n'a pas de sens.

Enfin, dans la bibliothèque ***directory.h***, la fonction ***isdirectory(...)*** est définie. Cette fonction retourne ***true*** si le chemin passé en argument désigne un (sous-)répertoire. Je vous invite à rechercher sur Internet de la documentation sur la macro ***S_ISDIR(...)***.

2 La bibliothèque

Ci-après figure le contenu de cette bibliothèque de parcours des entrées d'un répertoire. Bonne lecture.

```
1  // Fichier 'directory.h'
2  #include <string>
3  // Structures 'DIR' et 'dirent', opendir(...), readdir(...) et closedir(...).
4  #include <dirent.h>
5  // Structure 'stat', stat(...) et macro IS_DIR(...).
6  #include <sys/stat.h>
7
8  class Directory {
9    DIR * ptr_DIR_ ; // Pointeur sur une structure 'DIR', définie dans 'dirent.h'.
10 public :
11   // Unique constructeur autorisé ; 'opendir(...)' retourne le pointeur nul en cas
12   // d'erreur.
13   Directory (const std::string & chemin) : ptr_DIR_(opendir(chemin.c_str())) {}
14   // Destructeur.
15   ~Directory () {
16     if ( ptr_DIR_ != 0 ) {
```

```
17          closedir(ptr_DIR_) ; } }
18    // Test de l'échec : syntaxe 'if ( ! directory ) { // Traitement de l'erreur...'.
19    bool operator!() { return ptr_DIR_ == 0 ; }
20    // Déclaration de la fonction amie.
21    friend bool getentry(const Directory &, std::string &) ;
22  } ;
23  // Accès à l'entrée suivante du répertoire ; retourne 'false' à la fin ou en cas
24  // d'erreur.
25  bool
26  getentry(
27    const Directory & directory, // Argument en entrée, instance de Directory.
28    std::string & nom)           // Argument en sortie, nom de l'entrée suivante.
29  {
30    // Pointeur vers une structure 'dirent', définie dans 'dirent.h'.
31    dirent * ptr_entry = readdir(directory.ptr_DIR_) ;
32    // 'readdir(...)' retourne le pointeur nul à la fin ou en cas d'erreur.
33    if ( ptr_entry != 0 ) {
34      // 'd_name' est le membre de la structure 'dirent' de type 'char *'.
35      nom = ptr_entry->d_name ;
36      return true ; }
37    return false ;
38  }
39  bool
40  isdirectory(const std::string & chemin)
41  {
42    // Structure 'stat' définie dans '/sys/stat.h'. Il faut préfixer le nom du type par
43    // le mot 'struct' parce que le fichier '/sys/stat.h' donne aussi le nom 'stat' à
44    // une fonction.
45    struct stat ma_stat ;
46    // La fonction 'stat(...)' renseigne une structure 'stat' pour le nom passé en
47    // premier argument ; elle retourne 0 en cas d'échec.
48    if ( stat(chemin.c_str(), &ma_stat) != 0 ) {
49      return false ; }
50    // La macro 'S_ISDIR', appliquée au membre 'st_mode' d'une structure 'stat', teste
51    // s'il s'agit d'un directory.
52    return S_ISDIR(ma_stat.st_mode) ;
53  }
```