Documentation du module BINIO

Philippe WAILLE (UFR IMA, université Joseph Fourier)

Mai 2013

1 Introduction : séquences de bits et d'octets

1.1 Principe des entrées/sorties d'entiers en binaire

Dans binio, le contenu d'un fichier est considéré comme une séquence de SB bits. Cette séquence SB est la concaténation des représentations en binaire d'une suite d'entiers SE.

Chaque entier de SE a son propre format de représentation (nombre de bits, naturel ou relatif représenté en complément à 2). Des bits muets sont éventuellement ajoutés à la fin de SB pour obtenir un nombre entier d'octets dans le fichier.

L'utilisateur doit utiliser la même séquence de format d'entiers en écriture et en relecture. BINIO n'enregistre pas dans le fichier la longueur en bits de la séquence SB : il n'est donc pas possible à partir du seul contenu du fichier de déterminer combien de bits du dernier octet appartiennent à la séquence SB stockée dans le fichier.

Ce n'est pas un problème dans le cadre d'une méthode de compression de fichier sans perte (telle que Huffman ou LWZ) : il suffit d'ajouter un symbole spécial de fin au dictionaire des symboles à encoder.

1.2 Ordre des bits dans la suite d'entiers à lire ou écrire

Le paramètre booléen reverse_bit_order précise dans quel ordre les bits de l'entier sont ajoutés à la séquence de bits SB :

- 0 : le premier bit ajouté à SB est le bit de poids fort de l'entier, suivi par les autres bits par ordre décroissant de poids.
- 1 : le premier bit ajouté à SB est le bit de poids faible de l'entier, suivi par les autres bits par ordre croissant de poids.

1.3 Ordre de remplissage dans les octets dans le fichier

Le paramètre booléen pack_msb_first indique dans quel ordre les bits des octets du fichier sont parcourus pour représenter la séquence SB :

- 0 : du poids faible au poids fort : l'entier à écrire est cadré à droite de la partie encore libre de l'octet.

– 1 : du poids fort au poids faible : l'entier à écrire est cadré à gauche de l'espace libre dans l'octet.

La combinaison reverse_bit_order = 0, pack_msb_first = 1 est la plus facile à décoder pour un utilisateur humain. Dans ce cas, la séquence d'entiers en hexacécimal (5 sur 4 bits) (FE sur 8 bits) (3 sur 4 bits) donne la séquence de 2 octets suivante : 5F E3.

2 Accès aux fichiers

BINIO utilise un type binio_t qui permet d'identifier un flux binaire en cours de manipulation. Il est fourni par une fonction BINIO d'ouverture et utilisé par les fonctions BINIO de lecture ou écriture.

BINIO n'accède pas directement aux médias de stockage du flux binaire : cette tâche est déléguée à l'utilisateur.

Pour lire un flux binaire, BINIO utilise deux fonctions (eof_func, read_func ou write_func) fournies par l'utilisateur lors de l'ouverture du flux, qui utilisent un identificateur de support func_param de type pointeur.

Sur func_param, eof_func teste s'il reste (au moins) un octet à consommer et read_func lit un octet. Le principe est le même en écriture avec une fonction write_func d'écriture d'octet.

3 Initialisation et divers

BINIO peut ouvrir plusieurs flux simultanément. L'initialisation de BINIO en définit le nombre maximal. Un message de trace est envoyé sur la sortie standard d'erreur.

La fonction pending_bits retourne le nombre de bits pas encore traités de l'octet en cours.

4 Traces et débogage

Des mpessages d'information peuvent être générés sur la sortie standard d'erreur. Il suffit pour celà de définir et exporter une variable d'environnement :

export BINIO_DEBUG_LEVEL=2
La valeur de BINIO_DEBUG_LEVEL définit le degré de détail des traces générées.

5 Exemple

```
#include <stdlib.h>
#include "binio.h"

// Exemple d'utilisation des fonctions d'accès au contenu à copier
// pour lire la suite d'entiers à tranférer
// Principe similaire pour écrire la suite d'entiers à tranférer
```

```
#ifdef FROM_FILE

// Le contenu à traiter est dans un fichier

FILE *fichier_a_lire;

void initialiser (void)
{
  fichier_a_lire = fopen ("fichier_de_contenu","r");
  if (fichier_a_lire == NULL)
    {
    fprintf (stderr,
             "Echec de l'ouverture de fichier_de_contenu en lecture\n"
            );
    exit (EXIT_FAILURE);
    }
}

int my_eof (binio_file_func_param_t f)
{
  int lu;

  lu = getc (f);
  if (lu != EOF)
    {
    ungetc (lu,f);
    return 0;
    }
  else
    return 1;
}

#define my_get (binio_fgetc_t *) fgetc
#define my_param (binio_file_func_param_t) fichier_a_lire

#define MESSAGE "d'un fichier : fichier_de_contenu"

#else

// Le contenu à traiter est dans un tableau

char tableau_a_lire[] = "une chaîne terminee par 0";
int indice;

void initialiser (void)
{
  indice = 0;
}

bool fin_tableau (char t[])
{
```

```
  return  (indice >=0) && (t[indice] == 0);
}

int lire_tableau (char t[])
{
  int lu;
  indice++;
  lu = t[indice];
  return lu;
}

#define my_get (binio_fgetc_t *) lire_tableau
#define my_eof (binio_eof_t *) fin_tableau
#define my_param (binio_file_func_param_t) tableau_a_lire

#define MESSAGE "d'une chaine dans un tableau"

#endif

binio_t in;

void ouvrir (bool msb_first)
{
  initialiser ();
  printf ("my_param = %p, feof = %d\n", my_param, feof(my_param));
  in =  binio_read_open (my_get,
                         my_eof,
                         my_param,
                         1);
  if (in == NULL)
    {
    fprintf (stderr, "Erreur binio_read_open\n");
    exit (EXIT_FAILURE);
    }
}


// Pour simplifier, la fermeture du fichier de flux n'est pas geree

int main (void)
{
  binio_error_t errcode;
  unsigned long v;

  fprintf (stderr, "Lecture sur 4 bits du contenu %s\n",MESSAGE);

  ouvrir (1);
  while (1) {
    errcode = binio_unsigned_read (in,&v,4,0);
    switch (errcode) {
```

```
      case BINIO_OK : printf ("Lu sur 4 bits : %lx\n",v); break;

      case BINIO_EOF : printf ("Fin du contenu\n");
                       exit (EXIT_SUCCESS);

      default : fprintf (stderr,"Erreur lors de la lecture du flux binaire\n");
                exit (EXIT_FAILURE);

      }
  }
  return 0;
}
```