

Projet MSIDERE
Documentation

Description du projet	page 1
Documents utilisés	page 2
Partie I : Compréhension des données	page 2
Fichier sismo.dat	
Fichier sismo.cat	
Partie II : Travail effectué	page 8
Programmes écrits	
Problèmes rencontrés et solutions apportées	
Annexes	page 11

Description du projet :

Contexte général :

On dispose de stations qui détectent et enregistrent les mouvements du sol. Ces données sont ensuite transmises pour être étudiées et archivées. Les données ont été compressées dans un format « maison », cependant pour faciliter la transmission et le partage de ces données avec d'autres groupes d'étude des mouvements terrestres, ces données doivent être converties dans un format standard. Le format choisi est le format MiniSEED, variante du format SEED (Standard for the Exchange of Earthquake Data).

Type de travail :

Ce projet est donc de type conversion/ traitement de données. Il s'agit en effet de décompresser des données dans un format obsolète pour ensuite les mettre au format MiniSEED.

On dispose de fichiers sismo.dat et sismo.cat, l'un contenant des données compressées et l'autre contenant une sorte de table de lecture pour pouvoir comprendre le premier fichier. Le langage de programmation choisi est Python.

Existant :

Un programme en C qui permet de faire la conversion ainsi qu'un travail de lecture et décompression des données en Python.

Documents utilisés :

En cas de conflit d'interprétation, les deux premiers documents suivant doivent être pris comme documents de référence.

Document texte qui explique le format :
voir pièces jointes en fin, annexe 1

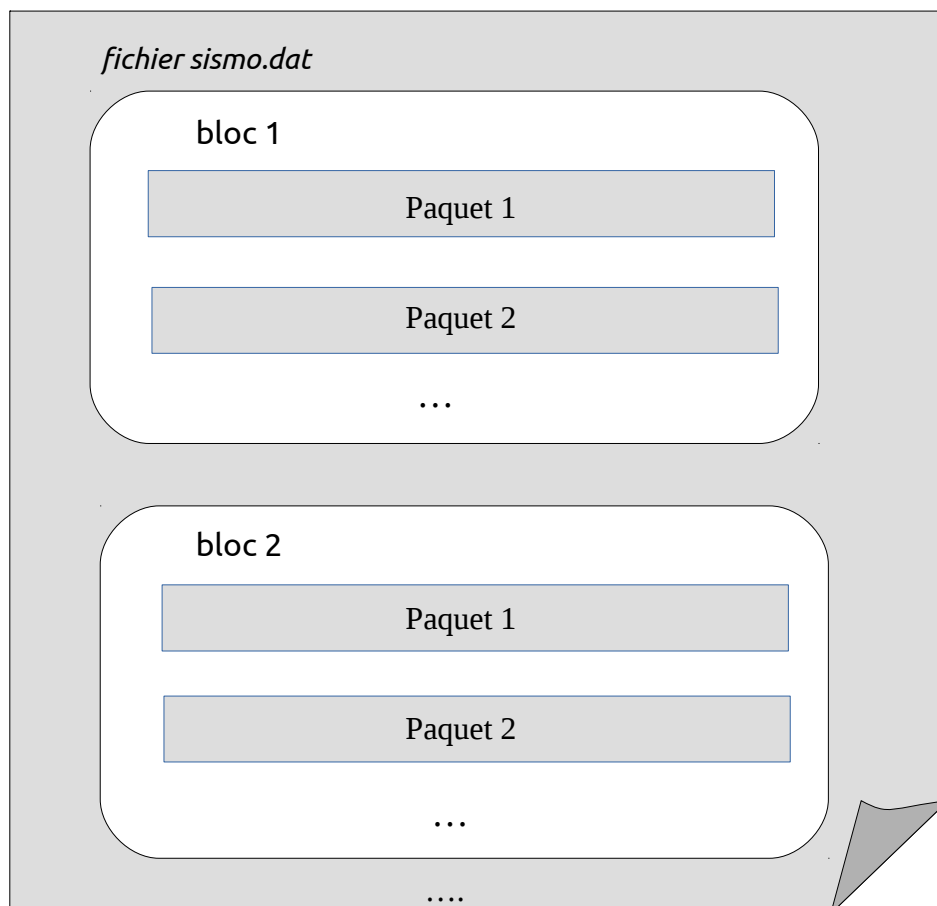
Programme en C :
voir pièces jointes en fin, annexe 2

Programme en Python

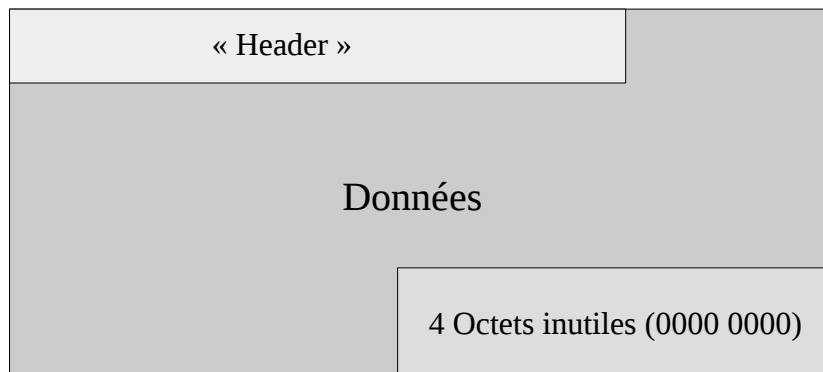
Partie I : Compréhension des données

Fichier sismo.dat:

La première partie du travail consiste à décompresser les données du fichier sismo.dat. Pour cela il est essentiel de bien comprendre comment ces données sont encodées. Chaque fichier sismo.dat est composé de blocs, eux mêmes composés de paquets selon le schéma suivant :



Chaque paquet commence par un header de 10 octets, suivis des données et de quatre octets inutilisés :



Constitution :

Chaque bloc est organisé de la manière suivante :

1. les **deux premiers octets** représente la taille du bloc
2. les octets suivants sont les paquets

Une fois qu'on a lu le nombre d'octets correspondant à la taille du bloc, on passe à un autre bloc constitué de la même façon.

Chaque paquet est constitué de la manière suivante :

1. les **dix premiers octets** constituent le header qui contient des informations sur le paquet
chaque « header » est composé de la manière suivante : (1)
 - a) les deux premiers octets représentent la taille du paquet, header compris
 - b) les deux suivants représentent le nombre d'échantillon du paquet (normalement égal à 128)
 - c) les deux suivants la première valeur
 - d) les deux suivants la composante continue
 - e) les deux derniers l'encodage sur n bits
2. les octets suivants sont les données compressées, sauf les 4 derniers octets qui sont « inutilés », il ne faut donc pas en tenir compte. On va dire qu'on peut les interpréter comme un signal de fin, même si cela n'est pas exact. (C'est un résidu malencontreux des premières versions qui étaient supposées gérer un nombre quelconque d'échantillons)

(1) Pour comprendre la signification de ces valeurs regarder « principe de la compression »

Une fois qu'on a lu le nombre d'octets correspondant à la taille du paquet, on passe à un autre paquet constitué de la même façon et ce jusqu'à la fin du bloc.

Principe de la compression :

Les données d'origine sont encodés sur 16 bits, mais ces 16 bits ne sont pas toujours utilisés. Les premiers bits sont donc généralement des zéros, par exemple :

échantillon 1 : 0000010110101011
 échantillon 2 : 0000000111011010

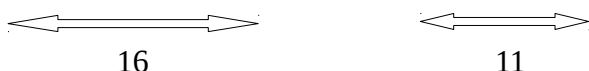
Tout d'abord, on passe tout en positif en ajoutant la valeur de l'échantillon la plus basse. Imaginons que l'on ai les valeurs suivante (cette fois-ci en décimal) :

[20, 12, -5, 6, -15, 20] → devient [35, 27, 10, 21, 0, 35] (on ajoute 15 à toutes les valeurs)

Cela permet de ne plus avoir à coder un bit de signe, et le 15 deviendra la « composante continue » du paquet. La « première valeur » est la première valeur du paquet, qui elle ne sera pas compressée, ce qui veut dire que la première valeur contenu dans les données sera en fait la deuxième valeur.

Le principe de la compression est de coder sur le maximum de bits occupé pour tout un paquet (un paquet correspond à 128 échantillon). Si on reprends l'exemple :

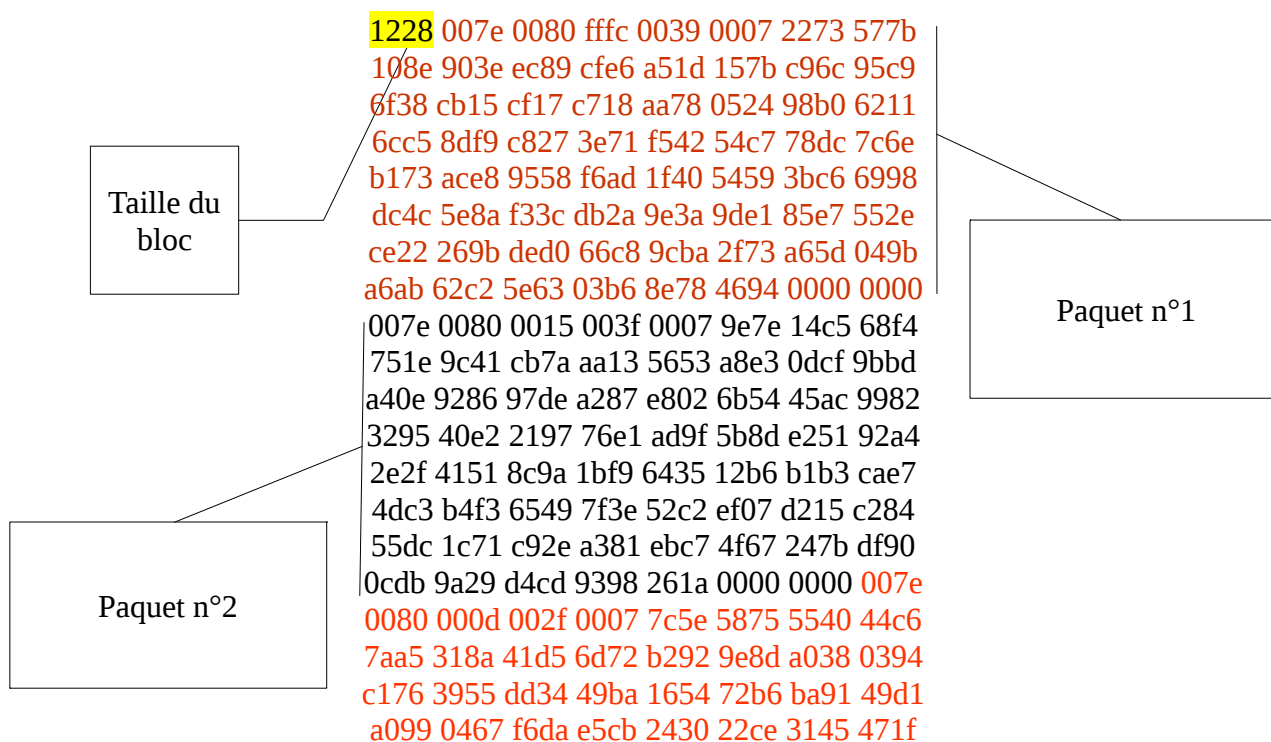
échantillon 1 : 0000010110101011 → devient 10110101011
 échantillon 2 : 0000000111011010 → devient 00111011010



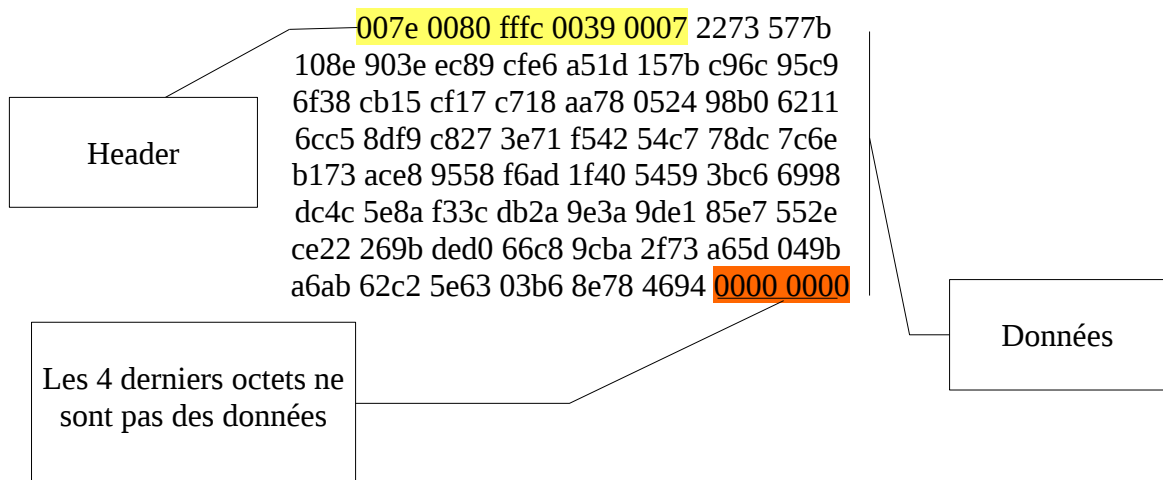
Le nombre de bits restant est contenu dans les deux derniers bits du « header ». C'est à dire qu'ici par exemple le « n » d'encodage sur n bits vaudra 11.

Exemple :

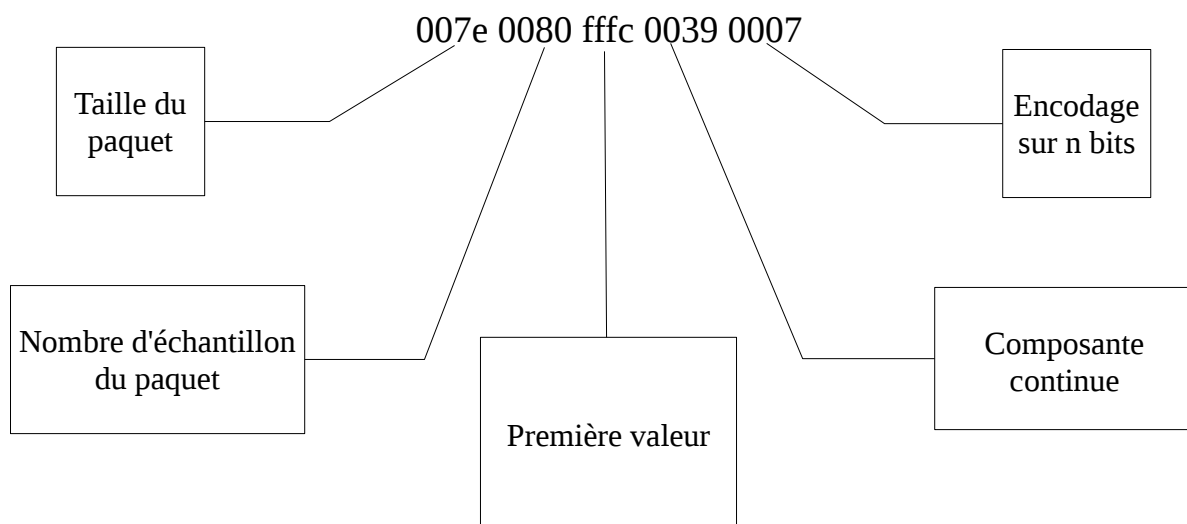
Sortie « standard » des vingt premières lignes d'un fichier sismo.dat en **hexadécimal** (obtenue en tapant dans la console : `od -x sismo.dat | head -n 20`) :



Analyse du premier paquet :



Analyse du « header » :



/!\ attention ces valeurs sont en hexadécimal, il faut donc les convertir avant de les exploiter /!\

Note :

A l'affichage, les octets peuvent apparaître inversés, il ne faut pas en tenir compte, quand on lis en binaire les fichiers, la machine les lis dans « le bon sens ». Ceci est du fait de l'endianness.

La taille du paquet peut être inférieure au nombre d'échantillon car les échantillons peuvent être codés sur moins de un octet.

Interprétation :

Les stations enregistrent les mouvements du sol sur plusieurs canaux, chacun correspondant à une donnée précise, certains canaux peuvent être non utilisés. Dans le cas des réseaux télémétrés

la description de la configuration du réseau se trouve dans le fichier config.txt qui accompagne les données. Chaque bloc représente une minute de signal d'un canal, ainsi dans le fichier sismo.dat on trouve codé :

```

minute 1 canal 1 ( bloc 1 )
minute 1 canal 2 ( bloc 2 )
minute 1 canal 3 ( bloc 3 )
...
minute 1 canal n ( bloc n )
minute 2 canal 1 ( bloc n + 1 )
minute 2 canal 2 ( bloc n + 2 )

```

Voilà pour l'essentiel du fichier sismo.dat.

Fichier sismo.cat :

La deuxième partie du projet, qui sera en fait la première à faire, consiste à déchiffrer le fichier sismo.cat dans lequel les informations essentielles à la compréhension de sismo.dat sont marquées.

Le fichier est organisé comme suit (*extrait de format_stras.txt*) :

Les 16 premiers octets du fichier sont mis à jour chaque minute:

- B0, B3:** offset où se fera la prochaine écriture dans sismo.dat.
- B4, B7:** " " sismo.cat.
- B8, B11:** offset de "bouclage" du fichier sismo.cat (non utilisé pour Géostar, seulement pour écriture "circulaire" réseau téléométré).
- B12,B13:** Code cadence échantillonnage pour Géostar. (Non utilisés pour réseau téléométré: information dans config.txt).
- B14,B15:** Numéro de la station Géostar. (" " " " ").

Chaque minute, 16 octets sont écrits/ajoutés dans le fichier:

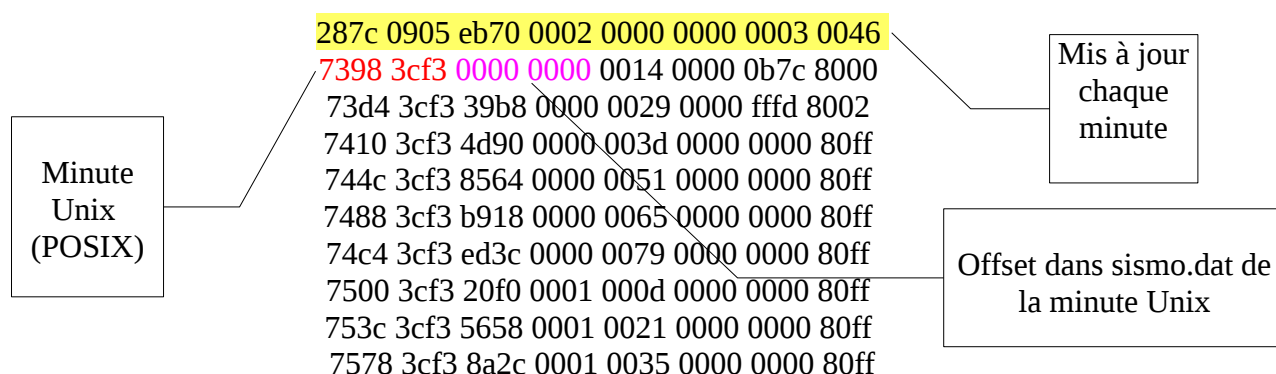
- B0, B3:** identification de la minute (Unix time_t).
- B4, B7:** offset dans sismo.dat des données correspondantes.
- B8, B9:** index de l'échantillon "seconde 0" dans le dernier bloc.
- B10, B11:** canaux en déclenchement dans la minute.
- B12, B13:** correction horloge (on ajoute/retranche un/des échantillons pour rester calé sur le top minute).
- B14 :** canal "actif" lors du top GPS (pour gestion fine de la dérive horloge Géostar).
- B15 :** valeur utilisée pour la correction de dérive thermique du quartz Géostar.

Description de la structure correspondante (en C) :

```
struct entete_cat
{
    time_t timepc;
    unsigned long point_dat;
    int decalage;
    int etat_trig;
    int cor_gps;
    unsigned char skew;
    unsigned char tcxo;
};
```

Exemple :

Sortie « standard » des dix premières lignes d'un fichier sismo.cat en **hexadécimal** (obtenue en tapant dans la console : `od -x sismo.cat | head`) :



La première ligne est mise à jour chaque minute au moment de l'écriture du fichier, au moment de sa lecture elle sera donc toujours la même.

Chaque ligne suivant la deuxième est constituée de la même manière.

L'offset inscrit correspond à l'offset de la minute Unix **canal 1**. C'est à dire que à l'offset (comprendre à l'octet n°) 0000 0000 on trouvera le bloc correspondant au canal 1 minute 7398 3cf3, à l'offset 0000 39b8 le bloc correspondant au canal 1 minute 73d4 3cf3, etc ... On peut ainsi en déduire la taille occupé par la minute.

Attention à l'endianness, les données affichés ici ne le sont pas forcément dans le bon sens !

/!\ attention /!\ :

Ce fichier comporte des informations qui ne sont pas entièrement comprises à l'heure de l'écriture elle ne sont pas représenter sur l'exemple mais vous pourrez trouver leurs positions en vous référant à la description du fichier donnée page précédente:

- l'index de l'échantillon seconde 0 dans le dernier bloc
- canaux en déclenchement dans la minute
- correction de l'horloge
- canal actif lors du top gps
- valeur utilisée pour la correction de dérive ...
- ainsi que le code cadence échantillonnage qui n'est pas encore connu.

Partie II : Travail effectué

1) Programmes écrits :

Au cours du projet trois script ont été écrits, deux l'ont essentiellement été pour procéder à des tests. Le troisième avait pour vocation d'être le programme rendu. Les pseudos codes n'ont été écrit que pour le dernier programme puisque les fonctions sont communes à plusieurs programmes. Dans tout ce qui suit l'offset est à comprendre comme la valeur de octet de l'endroit où on est, par exemple l'offset de début de bloc = 4650 signifie que le bloc commence à l'octet 4650.

msidere.py :

Fichier « test » , il est composé de deux fonctions et du main. La première fonction est commune aux trois programmes, elle permet de faire une liste des blocs du fichier sismo.dat. Dans msidere.py elle retourne pour chaque bloc l'offset de début de ce bloc et la taille du bloc .

La deuxième fonction est commune avec lesid.py, elle permet d'effectuer la décompression sur un bloc et retourne les données décompressées pour ce bloc.

Le main permet de lire un bloc (passé en argument) et de le décompresser.

regiseed.py :

regiseed.py est composé de 3 fonctions, la première liste les blocs du fichier .dat mais ne renvoie que l'offset de début de chaque bloc, la deuxième liste les paquets présents dans un bloc données (offset de début de bloc) , renvoyant les offset de début de chaque paquet. Cette fonction reprends le même principe de construction que la première, elle prends en argument le fichier sismo.dat et l'offset de début de paquet. La dernière renvoie les données et métadonnées d'un paquet sous forme d'un tuple.

lesid.py :

C'est le programme qui était destiné à effectuer la conversion. lesid est composé de 3 fonctions :

-La première renvoie la liste des blocs du fichier, elle prends en argument le fichier sismo.dat

Pseudo code :

liste vide

tant qu'on est pas à la fin du fichier :

lire ou on est (= offset de début de bloc)

lire deux octets (= taille du bloc)

ajouter à la liste l'offset de début de bloc

aller à la position actuelle + taille du bloc

fin tant que

retourner la liste

Cette fonction est présente dans les deux autres programmes

-La deuxième décompresse un bloc de données, elle prends en argument le fichier sismo.dat et l'offset de début du bloc

Pseudo code :

```

liste vide
on se place au début du bloc
on lis la taille du bloc
tant que taille du bloc > 0 :
    lis le header du paquet
    lis le nombre de bytes correspondant au données (sans les octets inutiles)
    si encodage sur n bits > 0 :
        convertis les données en binaire
        on coupe en morceaux de nbits
        on retranche la composante continue à chaque valeur
        on insère la première valeur
        on ajoute ce paquet à la liste
    on retranche la taille du paquet à la taille du bloc
on retourne les données décompressées (liste des valeurs)

```

-La troisième fonction permet de convertir les données décompressées en fichiers mseed elle est largement inspiré de la fonction de la doc obspy : « anything to MiniSeed », elle prends en argument les données à convertir, la channel correspondant et le temps « start time » de début des données

-Le main permet de décompresser et convertir en MiniSeed le nombre de minutes voulues

Pseudo code :

```

on ouvre le fichier .cat
on lis la première ligne
on lis n lignes n désignant le nombre de minutes passées en paramètre et on fais une liste des offset des minutes dans le fichier sismo.dat (ie chaque élément de la liste correspond à un offset du canal 1)

on ouvre le fichier sismo.dat
on fait la liste des blocs du fichier
on compte le nombre de canaux de la station (pour gérer les stations Géostar et les réseaux télémétrés, respectivement 4 et 8 ou 16 canaux)
pour toutes les minutes
    pour tous les canaux :
        on décomprime le bloc
        si il y a des données :
            on les convertis en MiniSeed
        sinon :
            on dit qu'il n'y a pas de données à convertir
    fin pour tout
fin pour tout

```

Les programme msidere.py et lesid.py ont aussi les fonctions nécessaire pour gérer les commentaires (module logging) et pour gérer les arguments passés directement en ligne de

commande (module argparse). Pour l'utilisation de ces programmes, rajouter `- - help` en ligne de commande (exemple « `./lesid.py - - help` ») pour obtenir des informations supplémentaire.

2) Problèmes rencontrés et solutions apportées:

Plusieurs problèmes ont rapidement été rencontrés :

La question de la décompression à été le premier et le plus dur d'entre eux. Celle ci est ardue puisqu'on doit faire des opérations bit à bit. Or si on peut facilement accéder à un octet avec python, il est difficile d'accéder aux bits. Les deux seules façon pour faire des opérations sur les bits sont les suivantes :

1. convertir en binaire grâce à la fonction `bin()`
 - avantages :
On voit mieux les choses, on travaille directement avec les bits et on peut facilement effectuer les opérations requises
 - inconvénients :
Nécessité de convertir d'abord la sortie en binaire en int pour remettre en binaire, car impossible de convertir les hexa en binaire directement. De plus, les valeurs peuvent être tronquée, en effet `01010110` sera renvoyé comme étant le binaire `1010110` et pour travailler sur la taille des bits cela peut être contraignant.
2. travailler avec les bitwise opérations
 - avantages :
On travaille au niveau des bits, beaucoup plus sûr
 - inconvénients :
Besoin d'une conversion en integer, un peu plus difficile pour un gain peu important au final

La solution choisie est la première, car la deuxième est plus compliqué et au final on ne gagne pas grand-chose. Pour limiter l'impact des bits tronqués, on se servira de la méthode `rjust()` qui permet de « justifier » les données en rajoutant les 0 qui ont été enlevés.

Pour lire les données en binaire, on a fait appel aux fonctions `pack` et `unpack` du module `struct` spécialement conçu pour gérer les struct telle qu'elle peuvent exister en C. Uniquement pour les données à décompressées, on a utilisé la fonction `hexlify` du module `binascii` qui lis un nombre quelconque de bytes, contrairement à `unpack` ou il faut renseigner le nombre de d'octets qu'on va lire. Quand on récupère les données à décompresser, on les lis d'un seul bloc pour ensuite les découper, ceci car lorsque l'encodage se fait par exemple sur 7 bits et que l'on lis par octet, il est difficile d'isoler ces 7 bits et de séparer le dernier, d'où la lecture par paquet entier.

Pour dater les échantillons, on se sert du fichier `sismo.cat`, mais comme toutes les infos n'ont pas été clairement comprises, la dérive observée n'est pas corrigée, elle se traduit par des « gaps » ou des « overlaps » lorsqu'on essaie de créer un fichier MiniSeed de plusieurs minutes.

Pour la composante continue de chaque paquet, elle a été comprise comme étant le minimum à ajouter pour que toutes les valeurs soient positives, or il s'avère que sur certains paquets ce ne soit pas le cas, cela peut venir d'une mauvaise compréhension de cette valeur, d'une mauvaise décompression des données ou simplement d'un mauvais encodage.

Conclusion :

Les programmes écrits permettent de se balader de bloc en bloc et de paquet en paquet assez facilement et de façon certaine. La lecture est décompression des données paraît exacte, mais puisque beaucoup d'informations ont dû être deviné, il se peut qu'elle ne le soit pas. Elle est en tout cas probablement non optimale. Le passage en MiniSeed est géré mais la gestion du temps est à compléter, la dérive n'étant pas corrigée. Attention pour le passage en MiniSeed, le sampling rate n'est pas dynamique, il faut le modifier à la main. Le programme lesid.py renvoie donc plusieurs fichiers au format MiniSeed, nommé en fonction de la date correspondante aux données et du canal.

Annexes 1:**Fichier format stras.txt :**

Les fichiers binaires sismo.cat et sismo.dat encodent les enregistrements continus des stations autonomes "Géostar" ou des réseaux télémétrés (8 ou 16 canaux).

Dans ce dernier cas, un fichier texte config.txt décrit la configuration du réseau (nombre de canaux, cadence d'échantillonnage, nom des canaux...).

Structure du fichier sismo.cat:

Chaque minute, 16 octets sont écrits/ajoutés dans le fichier:

B0, B3: identification de la minute (Unix time_t).

B4, B7: offset dans sismo.dat des données correspondantes.

B8, B9: index de l'échantillon "seconde 0" dans le dernier bloc.

B10, B11: canaux en déclenchement dans la minute.

B12, B13: correction horloge (on ajoute/retranche un/des échantillons pour rester calé sur le top minute).

B14 : canal "actif" lors du top GPS (pour gestion fine de la dérive horloge Géostar).

B15 : valeur utilisée pour la correction de dérive thermique du quartz Géostar.

Description de la structure correspondante:

```
struct entete_cat
{
    time_t timepc;
    unsigned long point_dat;
    int decalage;
    int etat_trig;
    int cor_gps;
    unsigned char skew;
    unsigned char tcxo;
}f;
```

et les 16 premiers octets du fichier sont mis à jour chaque minute:

B0, B3: offset où se fera la prochaine écriture dans sismo.dat.

B4, B7: " " sismo.cat.

B8, B11: offset de "bouclage" du fichier sismo.cat (non utilisé pour Géostar, seulement pour écriture "circulaire" réseau télémétré).

B12, B13: Code cadence échantillonnage pour Géostar. (Non utilisés

pour réseau télémétré: information dans config.txt).

B14,B15: Numéro de la station Géostar.

" " " ").

Structure du fichier sismo.dat:

Canal 1 minute 1, canal 2 minute 1, ... canal N minute 1, canal 1 minute 2, canal 2 minute 2, canal N minute 2, canal 1 minute 3 ,

la séquence des offsets du premier canal est écrite dans sismo.cat ("point_dat").

La taille de chaque bloc est encodée sur les 2 premiers octets, et on a ensuite une succession de paquets de 128 échantillons comprimés.

Le principe de la compression est d'encoder les différences successives avec le nombre de bits nécessaire à la plus grande de ces différences.

Ce nombre de bits, la taille du paquet ainsi que la valeur du premier échantillon sont écrits au début du paquet dans la structure:

```
struct entete_bloc
```

```
{
    short nbytes;           /*nb octets compactes          */
    short nech;             /*nb echantillons a restituer */
    short val0;             /*premiere valeur          */
    short offset;           /* "composante continue "    */
    short nbits;            /* encodage sur nbits        */
};
```

et la fonction de décompression:

```
short undelta(pack,data)           /* renvoie le nombre
d'echantillons */
```

```
    unsigned char *pack;           /* bloc de données compactées
*/
```

```
    short *data;                   /* zone où l'on restitue les données */
```

```
{
    unsigned short j,dispo,ntot,strip;
    unsigned long i,mask;
    struct entete_bloc *f;
    ntot=0;
    i=0;
    f=(struct entete *)pack;
    if ((f->nech!=128)|| (f->nbits>16)) {
        printf ("!");
        return (128);
    }
    if (f->nbits==0) { /*cas particulier canal 4 geostar: aucune
```

```

information à encoder quand GPS off */
    for (j=0;j<f->nech;j++)
        *(data+j)=f->val0;
    return (f->nech);
}
mask=0x80000000; /*msb =1 */
for (i=0;i<f->nbits-1;i++) {
    mask>>=1;
    mask=mask+0x80000000;
}
j=sizeof(*f);
    i= ((unsigned long)* (pack+j)<<24) +((unsigned
long)*(pack+j+1)<<16)+((unsigned long)*(pack+j+2)<<8)+(unsigned
long)*(pack+j+3);
    dispo=32;
    ntot=0;
    strip=0;
    while (ntot<f->nech) {
        while (dispo>=f->nbits) {
            *(data+ntot)=(unsigned short)((i & mask)>>(32-f->nbits));
            ntot ++;
            i<=&f->nbits;
            dispo-=f->nbits;
            strip+=f->nbits;
        }
        j=j+(strip/8); /* nombre d'octets traites completement */
        i= ((unsigned long)* (pack+j)<<24) +((unsigned
long)*(pack+j+1)<<16)+((unsigned long)*(pack+j+2)<<8)+(unsigned
long)*(pack+j+3);
        strip=strip%8;
        i<=&strip;
        dispo=32-strip;
    }
    /* on retranche la composante continue */
    for (i=0;i<f->nech;i++)
        *(data+i)-=f->offset;
    /* on restitue les valeurs initiales */
    *data=f->val0+*data;
    for (i=1;i<f->nech;i++) *(data+i)+=*(data+i-1);
    return (f->nech);
}

```

Annexes 2:

Fichier mseedifie1112.c :

```
/* Version "dump to mseed": on part des fichiers sismo.*  
pour générer nbcan fichiers miniseed sans passer par la  
phase extraction / génération arborescence  
On repère les blocs continus, on vérifie les corrections  
top gps, on les décomprime en  
mémoire et on les mseedifie */
```

```
/* Novembre 2011. Conversion en miniseed de  
fichiers Géostar, avec prise en compte dérive  
horloge / GPS */  
/* Conversion fichiers terrain "Strasbourg" en fichier  
* miniseed. Compression utilisée: Steim2.  
*  
* Chaque bloc de données occupe 4096 octets.  
* Les 64 premiers sont un header, on a ensuite  
* une série de 63 blocs de 64 octets.  
* Les 4 premiers octets de chaque bloc donnent,  
* 2 bits par 2 bits, le type de compression  
* utilisée:  
* 00=pas de données ou valeurs début, fin bloc  
* 01=4 valeurs encodées sur 8 bits  
* 10= regarder les 2 msb de chaque mot:  
*   01 = 1 valeur encodée sur 30 bits  
*   10 = 2 valeurs "      15  "  
*   11 = 3 valeurs sur 10 bits  
* 11= regarder les 2 msb de chaque mot:  
*   00 = 5 valeurs sur 6 bits  
*   01 = 6 valeurs sur 5 bits  
*   10 = 7 valeurs sur 4 bits  
*  
* Les valeurs brutes de début et de fin du bloc  
* sont données dans les second et troisième mot,  
* encodées sur 4 octets.  
*  
* */
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <string.h>  
#include <ctype.h>
```

```
#include <math.h>
#include <unistd.h>

struct btime
{
    unsigned short an;
    unsigned short jour_an;
    unsigned char heure;
    unsigned char minute;
    unsigned char sec;
    unsigned char bid;
    unsigned short dix_mill;
};

struct entete_seed
{
    char seq_num[6];
    char qualit;
    char space;
    char station[5];
    char loc[2];
    char ch[3];
    char netw[2];
    struct btime t;
    short nbsps;
    short sps;
    short fsps;
    char flag1;
    char flag2;
    char flag3;
    unsigned char nb_blocs;
    int time_corr;
    short data_begin;
    short first_blockette;
};

struct b1000
{
    unsigned short id;
    unsigned short next;
    unsigned char encode;
    unsigned char endian;
    unsigned char lg_exp2;
    unsigned char bid;
};

struct b1001
{
    unsigned short id;
    unsigned short next;
    unsigned char time_qual;
    unsigned char micro_sec;
```



```

        unsigned char reserved;
        unsigned char frame_count;
    };
struct bloc
{
    struct entete_seed head;
    struct b1000 b1000;
    struct b1001 b1001;
    unsigned int data[63][16];
};

```

/* renvoie le nb de bits minimum pour encoder val,"justifié steim2" */

```

int nbbits (int val)
{
    int n=31,place[7]={4,5,6,8,10,15,30},i;
    if (val<0)
        val*=-1;
    if (!val) return place[0];
    while (!(val & (1<<n--)));
    n+=3;
    i=0;
    while (n>place[i]) i++;
    return place[i];
}

```

/* cette fonction examine les entiers successifs à l'adresse d,
 * en tasse le plus possible (nb) dans l'entier qu'elle renvoie.
 */

```

unsigned int squeeze (int * d, int *nb,int dispo)
{
    unsigned pack=0,p;
    int nb_bits[7]={30,15,10,8,6,5,4};
    unsigned
    nib[7]={0x40000000,0x80000000,0xC0000000,0x00000000,0x00000000,0x4
    0000000,0x80000000};
    unsigned mask[7]={0x3FFFFFFF,0x7FFF,0x3FF,0xFF,0x3F,0x1F,0xF};
    int i,k,nb_fit,taille_cur,tot,occup[7];
    if (dispo<=0)
    {
        *nb=0;
        return 0;
    }
    if (dispo<7)
    {
        pack=d[0];

```

```
        pack &= mask[0];
        pack += nib[0];
        *nb = 1;
        return pack;
    }
    for (i=0; i<7; i++)
        occup[i] = nbbits(d[i]);
    nb_fit = 7;
    while(1)
    {
        taille_cur = occup[0];
        for (i=0; i<nb_fit; i++)
            if (occup[i] > taille_cur)
                taille_cur = occup[i];
        if (nb_fit * taille_cur > 32)
            nb_fit--;
        else break;
    }
    k = nb_fit;
    if (k == 4)
        k--; /* parade "bug 8 bits"... */
    for (i=0; i<k; i++)
    {
        pack <<= nb_bits[k-1];
        p = d[i] & mask[k-1];
        pack += p;
    }
    pack += nib[k-1];
    *nb = k;
    return pack;
}

void ecris_header_fixe (struct bloc * b)
{
    b->head.qualit = 'Q';
    b->head.space = ' ';
    b->head.loc[0] = 0;
    b->head.loc[1] = 0;
    b->head.data_begin = 64;
    b->head.first_blockette = 48;
    b->head.time_corr = 0;
    b->head.flag1 = 0; /* correction dans time_corr non appliquée */
    b->head.flag2 = 0x20; /* clock locked */
    b->head.flag3 = 0;
    b->head.nb_blocs = 1;
    b->b1000.lg_exp2 = 12;
    b->b1000.id = 1000;
    b->b1000.encode = 11;
    b->b1000.endian = 0;
}
```

```
b->b1001.id=1001;
}
```

```
/* fonction qui crée le tableau des corrections au 1/10000ème
   à partir du tableau des offsets des tops GPS
*/
```

```
struct spsvar
{
int align;
int nbech; /*taille du bloc (en echantillons */
double pps; /* sps calculé pour le bloc */
};
```

```
struct spsvar * corr (int * top, int nb, float sps)
{
int i,j,k=0,nbpts=0,fill=0,nbsec;
struct spsvar * skew;
for (i=1;i<nb;i++)
    if ((j=(top[i]-top[k])%(int)sps))
        {
            nbpts++; /* on définit un nouveau bloc */
            k=i;
        }
skew=(struct spsvar*) malloc ((nbpts+1)*sizeof(struct spsvar));
k=0;
for (i=1;i<nb;i++)
    if ((j=(top[i]-top[k])%(int)sps))
        {
            nbsec=(top[i]-top[k]+50)/(int)sps;
            skew[fill].nbech=top[i]-top[k];
            if (!fill)
                skew[fill].nbech+=top[0];
            skew[fill++].pps=(double)(top[i]-top[k])/(double)nbsec;
            k=i;
        }
skew[fill].nbech=0; /* fin de tableau */
skew[fill].pps=(double)sps;
return skew;
}
```

```
#define NPAC 128
```

```
double restit (int nb, struct spsvar *tab)
```

```

{
double t=0.;
int i=0;
while ((nb>tab[i].nbech) && (tab[i].nbech))
{
t+=(double)tab[i].nbech/tab[i].pps;
nb-=tab[i].nbech;
i++;
}
t+=(double)nb/tab[i].pps;
return t;
}

short undelta(pack,data)          /* renvoie le nombre d'echantillons */
    unsigned char *pack;          /* bloc de donn-es compact-es */
    short *data;
    /* zone o  l'on restitue les donn-es */
{
struct entete
{
short nbytes;          /*nb octets compactes */
short nech;           /*nb echantillons a restituer */
short val0;           /*premiere valeur */
short offset;         /* "composante continue " */
short nbits;          /* encodage sur nbits */
}*f;
unsigned short j,dispo,ntot,strip;
unsigned long i,mask;
ntot=0;
i=0;
f=(struct entete *)pack;
if ((f->nech!=128)||((f->nbits>18))
{
printf ("!");
return (128);
}
if (f->nbits==0)
{
for (j=0;j<f->nech;j++)
*(data+j)=f->val0;
return (f->nech);
}
fflush (stdout);
mask=0x80000000; /*msb =1 */
for (i=0;i<f->nbits-1;i++)
{
mask>>=1;
mask=mask+0x80000000;
}
}

```

```

        j=sizeof(*f);
        i= ((unsigned long)* (pack+j)<<24) +((unsigned
long)*(pack+j+1)<<16)+((unsigned      long)*(pack+j+2)<<8)+(unsigned
long)*(pack+j+3);
        dispo=32;
        ntot=0;
        strip=0;
        while (ntot<f->nech)
        {
            while (dispo>=f->nbits)
            {
                *(data+ntot)=(unsigned short)((i & mask)>>(32-f->nbits));
                ntot ++;
                i<=&f->nbits;
                dispo-=f->nbits;
                strip+=f->nbits;
            }
            j=j+(strip/8); /* nombre d'octets traites completement */
            i= ((unsigned long)* (pack+j)<<24) +((unsigned
long)*(pack+j+1)<<16)+((unsigned      long)*(pack+j+2)<<8)+(unsigned
long)*(pack+j+3);
            strip=strip%8;
            i<=&strip;
            dispo=32-strip;
        }
        /* on retranche la composante continue */
        for (i=0;i<f->nech;i++)
            *(data+i)-=f->offset;
        /* on restitue les valeurs initiales */
        *data=f->val0+*data;
        for (i=1;i<f->nech;i++) *(data+i)+=*(data+i-1);
        return (f->nech);
    }

```

```

void affiche (short * ram, int nb, short v0)
{
    int i=0,oldval=v0,k=0;
    while (i<nb)
    {
        k++;
        if (ram[i]!=oldval)
        {
            printf ("%d fois %d\n",k,oldval);
            oldval=ram[i];
            k=0;
        }
        i++;
    }
}

```

```

    }
    printf ("%d fois %d\n",k,oldval);

}

```

```

int top (short *t, int n,int on,int off,float sps)
/* renvoie 1 si on a une bonne transition à l'indice n ET à l'indice n+sps */
{
    if (((t[n])==off) && ((t[n+1])==on))
        if (((t[n+(int)sps])==off) && ((t[n+1+(int)sps])==on))
            if (((t[n+2*(int)sps])==off) && ((t[n+1+2*(int)sps])==on))
                return 1;
    return 0;
}

```

```

int main(int argc, char * argv[])
{
    struct fcb
    {
        time_t timepc;
        unsigned long point_dat;
        short decalage; /* nb samples between block begin and second 0 */
        short etat_trig; /* trigger state of every channel in the minute */
        short cor;
        unsigned char canal;
        unsigned char vtcxo; /* available */
    }tmp;
    struct bloc_contig
    {
        int nsamples;
        time_t bloc_deb,bloc_fin;
        int duree;
    };
    struct bloc_contig *morceau;
    int ok=0,nblocs=0,shift,oldecal,spy_min,spy_max;
    float sps=75.;
    double t0,*t0bloc,offmoy,offnow,*corr_t0;
    int i,j,ii,ibis,k,nbcan=4,max_bloc,succes=0,nsamples,pas;
    int
    source,dest_ram,incr,old_offset,*indice_top,topon=2,topoff=3,nbtops,*d_ram;
    struct spsvar ** bid;
    short jshort;
    FILE *cat,*dat;
    unsigned char * buffer;
    short * signal_ram;
    short ** s_ram;
    int * int_ram;
}

```

```

double oldstable=0;
time_t oldtime,debut,fin;
/*****/
FILE *chan2disk;
int nb,nb_records,flag,nibble,oldval;
struct tm *tm,*t;
time_t timepc;
int old_index,index,seq_num=0;
double tcur;
struct bloc *bloc;
char *station[32]={"Z","N_S","E_W","Time"};
char titre_fich[128],id[8];
char **conf_sta, **conf_netwk, **conf_cmp;
bloc=(struct bloc *) malloc (sizeof(struct bloc));
/*****/
if (argc==1)
{
    if ((cat=fopen("sismo.cat","rb"))==NULL)
    {
        printf ("\n Fichier sismo.cat introuvable...");
        return 1;
    }
}
else
    if ((cat=fopen(argv[1],"r"))==NULL)
    {
        printf ("\n Fichier %s inaccessible...",argv[1]);
        return 1;
    }
argv[1][strlen(argv[1])-3]++;
    if ((dat=fopen(argv[1],"r"))==NULL)
    {
        printf ("\n Fichier %s inaccessible...",argv[1]);
        return 1;
    }
if (argc==3)
    sscanf (argv[2],"%d",&pas);
else pas=60;
shift=((int)sps*60)%NPAC;
putenv("TZ=XXX0"); /*no offset between PC clock and UTC:Unknown time zone,
no Daylight Saving Time*/
tzset();
    if      ((buffer=      (unsigned      char      *)      malloc
(64*(short)sps*2*sizeof(char)))==NULL) exit (1);
    max_bloc=(unsigned short) (sps*120.+1024);/* worst case: two bytes per
sample plus headers */
    signal_ram=(short *) malloc (max_bloc*sizeof(char));
if(fread (&tmp,16,1,cat)!=1)
{

```

```

        printf ("Fichier incorrect ou inexistant\n");
        fclose (cat);
        return 1;
    }
    printf ("%s\n",argv[1]);
    fread (&tmp,16,1,cat);
    oldtime=tmp.timepc;
    debut=oldtime;
    oldecal=tmp.decalage;
    while (fread (&tmp,16,1,cat)==1)
    {
        if ((tmp.timepc-oldtime==60L)&&(tmp.timepc%86400))
        {
            oldecal=tmp.decalage;
            oldtime=tmp.timepc;
            ok++;
        }
        else
        {
            nblocs++;
            fin=oldtime;
            ok=0;
            oldtime=tmp.timepc;
            debut=oldtime;
        }
    }
    fin=oldtime;
    nblocs++;
    /*printf ("\n Bloc %d:",++nblocs);*/
    /*printf ("\n %6d minutes cons-cutives OK de ",ok);
    t=gmtime(&debut);
        printf ("%04d/%02d/%02d      %02d:%02d\n",t->tm_year+1900,t-
    >tm_mon+1,t->tm_mday,t->tm_hour,t->tm_min);
    t=gmtime(&fin);
        printf ("\t\t\t\tA:      %04d/%02d/%02d      %02d:%02d\n",t-
    >tm_year+1900,t->tm_mon+1,t->tm_mday,t->tm_hour,t->tm_min);*/
    if ((morceau=(struct      bloc_contig      *)malloc      (nblocs*sizeof(struct
    bloc_contig)))==NULL)
    {
        printf ("Erreur alloc: trop de morceaux. Abandon...");
        fclose (cat);
        return 1;
    }
    if ((t0bloc=(double *) malloc (nblocs*sizeof(double)))==NULL)
    {
        printf ("Erreur alloc\n");
        return 1;
    }
    if ((corr_t0=(double *) malloc (nblocs*sizeof(double)))==NULL)

```



```

    {
        printf ("Erreur alloc\n");
        return 1;
    }
if ((bid=(struct spsvar **) malloc (nblocs*sizeof(struct spsvar *)))==NULL)
    {
        printf ("Erreur alloc\n");
        return 1;
    }
for (i=0;i<nblocs;i++)
    t0bloc[i]=0.;
nblocs=0;
fseek (cat,0,SEEK_SET);
fread (&tmp,16,1,cat);
fread (&tmp,16,1,cat);
oldtime=tmp.timepc;
debut=oldtime;
ok=0;
while (fread (&tmp,16,1,cat)==1)
    {
        if ((tmp.timepc-oldtime==60L)&&(tmp.timepc%86400))
            {
                oldtime=tmp.timepc;
                ok++;
            }
        else
            {
                morceau[nblocs].bloc_deb=debut;
                fin=oldtime;
                morceau[nblocs].bloc_fin=fin;
                morceau[nblocs++].duree=ok;
                ok=0;
                oldtime=tmp.timepc;
                debut=oldtime;
            }
    }
fin=oldtime;
morceau[nblocs].bloc_deb=debut;
morceau[nblocs].bloc_fin=fin;
morceau[nblocs++].duree=ok;
fseek (cat,16,SEEK_SET);
for (i=0;i<nblocs;i++)
    {
        j=morceau[i].duree;
        indice_top=(int *) malloc (j*sizeof(int));
        for (ii=0;ii<j;ii++)
            indice_top[ii]=0;
        nbtops=0;
        oldecal=0;
    }

```

```

old_offset=0;
/*printf ("Taille:%d\n",morceau[i].duree);*/
for (j=0;j<morceau[i].duree;j++)
{
    fread (&tmp,sizeof(struct fcb),1,cat);
    fseek (dat,tmp.point_dat,SEEK_SET);
    for (ibis=0;ibis<nbcab;ibis++)
    {
        source=0; /* offset dans le buffer de chaque sous-bloc trame*/
        dest_ram=0;
        fread (&jshort,2,1,dat);/* lecture nb octets du bloc minute*/
        fread (buffer,1,jshort,dat);
        if (ibis==3)
            while (source<jshort)
            {
                dest_ram+=undelta(buffer+source,signal_ram+dest_ram);
                incr=*(buffer+source)+(*(buffer+source+1)<<8);
                source+=incr;
            }
        t0=(double)tmp.timepc-(double)(old_offset)/sps+(double)(NPAC-
oldecal)/sps;
        if (tmp.canal!=0xff)
        {
            for (ii=0;ii<dest_ram;ii++)
            if (top(signal_ram,ii,topon,topoff,sps))
            {
                indice_top[nbtops++]=old_offset+ii;
                break;
            }
        }

        old_offset+=dest_ram;
        if (t0!=oldstable)
        {
            oldstable=t0;
            succes=0;
        }
        else
        {
            succes++;
            if (succes==4)
                if (t0bloc[i]==0.)
                    t0bloc[i]=t0;
        }
        oldecal=tmp.decalage;
    }
    bid[i]=corr(indice_top,nbtops,sps);
    free (indice_top);

```

```

    }
    fseek (cat,16,SEEK_SET);
    for (i=0;i<nblocs;i++)
    {
        offmoy=0.;
        j=morceau[i].duree;
        nbtops=0;
        oldecal=0;
        old_offset=0;
        nsamples=0;
        for (j=0;j<morceau[i].duree;j++)
        {
            fread (&tmp,sizeof(struct fcb),1,cat);
            fseek (dat,tmp.point_dat,SEEK_SET);
            for (ibis=0;ibis<nbcan;ibis++)
            {
                source=0; /* offset dans le buffer de chaque sous-bloc trame*/
                dest_ram=0;
                fread (&jshort,2,1,dat);/* lecture nb octets du bloc minute*/
                fread (buffer,1,jshort,dat);
                if (ibis==3)
                {
                    while (source<jshort)
                    {
                        dest_ram+=undelta(buffer+source,signal_ram+dest_ram);
                        incr=*(buffer+source)+(*(buffer+source+1)<<8);
                        source+=incr;
                    }
                    nsamples+=dest_ram;
                }
            }
            if (tmp.canal!=0xff)
            {
                for (ii=0;ii<dest_ram;ii++)
                if (top(signal_ram,ii,topon,topoff,sps))
                {
                    nbtops++;
                    t0=t0bloc[i]+restit(old_offset+ii,bid[i]);
                    offnow=t0-(int)t0;
                    if (offnow<.1) offnow+=1.;
                    offmoy+=offnow;
                    break;
                }
            }

            old_offset+=dest_ram;
            oldecal=tmp.decalage;
        }
        corr_t0[i]=1.-offmoy/nbtops;
    }

```

```

    morceau[i].nsamples=nsamples;
/* fin construction correction horloge */
}
ecris_header_fixe(bloc);
bloc->head.sps=(int)sps;
bloc->head.fsps=1;
for (i=0;i<nblocs;i++)
    t0bloc[i]=corr_t0[i];
fseek (cat,16,SEEK_SET);
d_ram=(int *) malloc (nbcan*sizeof(int));
s_ram=(short **) malloc (nbcan*sizeof(short *));
for (i=0;i<nblocs;i++)
{
    j=morceau[i].duree;
    if (j<60)
    {
        do
        fread (&tmp,sizeof(struct fcb),1,cat);
        while (tmp.timepc<=morceau[i].bloc_fin);
        /*fseek pour sauter au bloc suivant */
        continue;
    }
    int_ram=(int *) malloc (morceau[i].nsamples*sizeof(int));
    for (k=0;k<nbcan;k++)
    {
        if ((s_ram[k]=(short *) malloc
(morceau[i].nsamples*sizeof(short)))==NULL)
        {
            printf ("Erreur alloc canal %d\n",k);
            return 0;
        }
        d_ram[k]=0;
    }
    for (j=0;j<morceau[i].duree;j++)
    {
        fread (&tmp,sizeof(struct fcb),1,cat);
        fseek (dat,tmp.point_dat,SEEK_SET);
        for (ibis=0;ibis<nbcan;ibis++)
        {
            source=0; /* offset dans le buffer de chaque sous-bloc
trame*/
            fread (&jshort,2,1,dat);/* lecture nb octets du bloc minute*/
            fread (buffer,1,jshort,dat);
/*
            if (ibis-3)*/
            while (source<jshort)
            {
                d_ram[ibis]+=undelta(buffer+source,s_ram[ibis]
+d_ram[ibis]);
                incr=*(buffer+source)+(*(buffer+source+1)<<8);

```

```

        source+=incr;
    }
}
for (ibis=0;ibis<nbcanc;ibis++)
{
    index=0;
    old_index=0;
    tcur=t0bloc[i];
    t=gmtime(&morceau[i].bloc_deb);
    sprintf (titre_fich,"%4d_%02d_%02d_%02d_%02d.%s",t-
>tm_year+1900,t->tm_mon+1,t->tm_mday,t->tm_hour,t-
>tm_min,station[ibis]);
    chan2disk=fopen (titre_fich,"wb");
    printf ("Creation %s\n",titre_fich);
    fflush(stdout);
    for (j=0;j<morceau[i].nsamples;j++)
        int_ram[j]=(int)s_ram[ibis][j];
    k=int_ram[0];
    for (j=0;j<morceau[i].nsamples;j++)
    {
        oldval=int_ram[j];
        int_ram[j]-=k;
        k=oldval;
    }
    do
    {
        sprintf (id,"%06d",seq_num++);
        memcpy (&bloc->head.seq_num,id,6);
        timepc=(time_t)tcur;
        tm=gmtime(&timepc);
        bloc->head.t.an=tm->tm_year+1900;
        bloc->head.t.jour_an=tm->tm_yday+1;
        bloc->head.t.heure=tm->tm_hour;
        bloc->head.t.minute=tm->tm_min;
        bloc->head.t.sec=tm->tm_sec;
        bloc->head.t.dix_mill=(short)(10000.*(tcur-(int)tcur));
/*      bloc->head.netw[0]=conf_netwk[ibis][0];
        bloc->head.netw[1]=conf_netwk[ibis][1];
        my_strcpy (bloc->head.ch,conf_cmp[ibis],3);
        my_strcpy (bloc->head.station,conf_sta[ibis],4);*/
        flag=0;
        fin=0;
        for (j=0;j<63;j++)
            for (k=0;k<16;k++)
                bloc->data[j][k]=0;
        bloc->data[0][1]=(int)s_ram[ibis][index];
        for (j=0;j<63;j++)
        {

```

```

        nibble=0;
        for (k=(flag)?1:3;k<16;k++)
        {
            if (!flag)
                flag=1;
            nibble<=2;
            bloc->data[j][k]=squeeze
(int_ram+index,&nb,morceau[i].nsamples-index);
            index+=nb;
            switch (nb)
            {
                case 1:
                case 2:
                case 3:
                    nibble+=2;
                    break;
                case 4:
                    nibble+=1;
                    break;
                case 5:
                case 6:
                case 7:
                    nibble+=3;
                    break;
                default:break;
            }
            if (!nb)
                fin=1;
        }
        bloc->data[j][0]=nibble;
        if (fin) break;
    }
    bloc->head.nbsps=index-old_index;
    bloc->data[0][2]=(int)s_ram[ibis][index-1];
    fwrite (bloc,sizeof(struct bloc),1,chan2disk);
    nb_records++;
    tcur=t0bloc[i]+restit(index,bid[i]);
    old_index=index;
}
while (index<morceau[i].nsamples);
fclose (chan2disk);
}
for (k=0;k<nbcas;k++)
    free (s_ram[k]);
free (int_ram);
printf ("Fin traitement bloc %d\n",i);
fflush(stdout);
}
fclose (cat);

```

```
fclose (dat);  
return 0;  
}
```