TPBD

Très Petite Base de Données
Programmée par Mathias Kende

Table des Matières

TPBD *- 1 -*

Table des Matières **- 2 -**

Introduction **- 3 -**
Licence - 3 -
Installation - 3 -
Utilisation - 3 -
Remarque - 3 -
Me contacter - 3 -

Fonctionnement **- 4 -**
Préambule - 4 -
Philosophie - 4 -
Contraintes - 4 -
Possibilités - 5 -
Fonctions - 5 -
Gestion des erreurs - 5 -

Fonctions **- 6 -**
DBExist - 6 -
CreateDB - 6 -
OpenDB - 6 -
UpdateDB - 7 -
CloseDB - 7 -
DeleteDB - 7 -
CompressDB - 8 -
RecordExist - 8 -
CreateRecord - 8 -
DeleteRecord - 8 -
OpenRecord - 9 -
WriteRecord - 9 -
ReadRecord - 10 -
SeekRecord - 10 -
CloseRecord - 11 -
ResizeRecord - 11 -
GetRecordInfo - 11 -
UpdateRecord - 11 -
GetNumberOfEmptyRecord - 11 -
GetFirstEmptyRecord - 11 -
GetErrorString - 12 -

Constantes **- 13 -**
Numéro d’erreur - 13 -

Introduction

Licence

Ce logiciel est diffusé sous la licence SectionPC. Vous devez en avoir obtenue une copie en téléchargeant la bibliothèque. Si ce n'est pas le cas, vous pouvez la télécharger là : http://lib.sectionpc.info/licence/licence.html.

L'utilisation de ce logiciel entraîne l'acceptation de la licence.

Installation

Pour installer la bibliothèque, copiez tout simplement le fichier TPBD.h dans le répertoire « include » de votre compilateur et les fichier de la bibliothèque (TPBD.lib et TPBD_dll.lib pour Visual C++ et TPBD.a seulement pour Dev C++) dans le répertoire « lib » de votre compilateur.

Utilisation

Pour utiliser la bibliothèque, vous devez inclure l'en-tête TPBD.h dans vos projets. Sous Visual C++ rien d'autre n'est requis, mais avec Dev C++ vous devez spécifier à l'éditeur de lien d'utiliser le fichier TPBD.a. Pour cela, dans les options du projet (alt+P) vous devez allez soit dans l'onglet paramètre soit sur la page principale selon la version du logiciel, et à coté de la mention « éditeur de lien » cliquer sur le bouton « rajouter un fichier » puis parcourir l'arborescence pour ajouter le fichier TPBD.a.

Remarque

La bibliothèque n'en est qu'à ses débuts. Même si pour l'instant je nais pas remarqué de bogues cela e veut pas dire qu'elle en soit exempt (pour autant qu'un programme puisse l'être). Je vous conseille donc de ne pas l'utiliser dans un programme dont les données sont vitales. On me pardonnera aussi le fait que les fonctions de manipulations soient basiques car je n'ais pas encore eu le temps d'en améliorer la qualité.

Me contacter

Si vous avez des remarques positives ou négatives, des suggestions d'améliorations, des rapports de bogues ou n'importe quoi d'autre, vous pouvez m'écrire à : sectionpc@gmail.com.

Fonctionnement

Préambule

Cette bibliothèque est écrite en C++ bien qu'elle ne fasse pas appel au concept de classe. Néanmoins elle repose sur le concept des exceptions : il n'est pour l'instant pas possible d'obtenir des codes de retour pour les fonctions de la bibliothèque que vous appelez, à la place vous devez gérer les exceptions qui pourrait être levé par la bibliothèque (en théorie de telles exceptions ne se produisent pas si vous ne faites pas d'erreur dans l'utilisation des fonctions de la bibliothèque).

Toutes les exceptions levées sont de type entier et des constantes dans le fichier TPBD.h définissent les erreurs qui peuvent être levées. Ainsi, le code d'ouverture d'une base de donnée pourrait ressembler à cela :

```
try
{
	db=OpenDB("base_de_donnees.dat");
}
catch (int err_num)
{
	switch (err_num)
	{
		case ERROR_COULD_NOT_OPEN_FILE:
			cout << "Erreur : le fichier n'a pas pu être ouvert.\n";
			break;
		case ERROR_DB_ALREADY_OPENED:
			cout << "Erreur : une base de données est déjà ouverte.\n";
			break;
		default:
			cout << "Erreur n°" << err_num << ".\n";
	}
}
```

Bien sûr, en réalité vous ne devez gérer les exceptions que pour des groupes d'instructions et non pas pour chacune d'entre elle ce qui fait le code à écrire en plus des appels aux fonctions eux même est assez réduit (seul les opérations d'ouvertures et de créations de base de données risque réellement d'échouer si les fichiers ne sont pas disponible).

Philosophie

Les fonctions exportées par la bibliothèque ne sont pas sans rappeler celle de stdio concernant la manipulation des fichiers (fopen, fread et fwrite) et ce n'est pas un hasard. Car une fois que vous aurez ouvert une base de données, vous pouvez ouvrir chacun de ces enregistrements et récupérer un objet dont vous n'aurez pas à vous soucier mais qui a exactement le même rôle qu'un descripteur de fichier. Vous pourrez ensuite lire et écrire de manière séquentielle les enregistrements de la base de données.

Contraintes

A l'origine de ce projet ce trouve un moteur très simple, simpliste même, de gestion de base de données : j'avais un projet pour lequel je devais sauvegarder des données binaires dont le nombre m'était connu d'avance. De plus je connaissais la taille que ferait chaque enregistrement dans le fichier. Pour enregistrer les données, il suffisait alors d'écrire dans un fichier à une adresse bien déterminée pour chaque enregistrement. Mais très vite le nombre de données à enregistré peut grandir si bien que prévoir la place pour chaque enregistrement se révèle être trop coûteux en mémoire, ou bien la taille des enregistrement peut ne plus être connu d'avance, si bien que l'on ne peut plus calculer leurs position dans le fichier. C'est pour résoudre un tel problème qu'est née TPBD.

En résumé pour pouvoir utiliser le moteur TPBD, vous devez être capable de numéroter vos enregistrements dans la base de données et de prévoir à l'avance le nombre maximal

d’enregistrements que vous ferez (vous pouvez néanmoins largement surestimer ce nombre sans que cela ne vous coûte beaucoup de mémoire).

Possibilités

Dans sa dernière version, la bibliothèque permet d’ouvrir plusieurs bases de données en même temps. Et elle supporte mieux le multithreading, même si certaines fonctions ne peuvent toujours pas être appelées dans deux thread parallèle.

Fonctions

Toutes les fonctionnalités de la bibliothèque sont exportées aux moyens de fonctions. Dans le chapitre suivant, je décris chacune des fonctions de la bibliothèque dans l’ordre ou vous aurez à vous en servir (approximativement).

Gestion des erreurs

Comme indiqué au dessus, la bibliothèque utilise le mécanisme d’erreur du C++. Dans le descriptif de chaque fonction ci-après, je décris (de manière non exhaustive) les causes diverses qui peuvent faire échouer l’appel à une fonction. Mais il ne faut pas se formaliser, les seules possibilité réelle d’erreur proviennent de l’ouverture de la base de données comme je l’ais déjà dit, si le fichier n’existe pas ou s’il est ouvert par un autre programme. Et pour les opérations de lectures/écritures courantes, des erreurs ne peuvent se produire que s’il y a des erreurs au niveau de la logique du programme (écriture dans un enregistrement qui n’est pas ouvert, ou au-delà de sa taille maximal, etc.). Bien entendu, je ne peut pas exclure que des erreurs indépendantes aussi bien de votre code que du miens puisse se produire (erreur en écriture du au système qui corromprait la table d’allocation de la base de données par exemple), mais en réalité c’est assez peu courant.

Comme je ne peut tout de même pas exclure une erreur dans mon code (surtout du fait qu’il soit assez récent), si jamais vous rencontrer une erreur dont vous ne comprenez pas l’origine, essayez de noter le plus d’information susceptible de m’aider (ce qui est parfait dans ce cas là est un morceau de code qui permet de reproduire l’erreur en question) et envoyer le moi (Cf. ci-dessus pour mon adresse). J’essayerais de corriger le problème dans les plus brefs délais.

Fonctions

DBExist

Indique si un fichier est ou non une base de données.

```
int DBExist(char* DB);
```

Paramètre

DB : pointeur vers une chaîne de caractères terminée par un zéro qui indique le nom du fichier sur lequel on veut des informations.

Valeur de retour

Renvoie FILE_DOES_NOT_EXIST si le fichier n'existe pas, FILE_IS_NOT_DB si le fichier existe mais qu'il ne s'agit pas d'une base de données et FILE_IS_DB si le fichier existe et qu'il s'agit d'une base de données.

Informations

Cette fonction ne peut pas échouer (mis à pars en cas d'erreurs en lecture qui seraient du au système). Mais les informations qu'elle retourne sont à utiliser avec précautions : Si la fonction indique que le fichier n'existe pas, cela peut vouloir dire qu'il est actuellement ouvert par un autre programme qui aurait verrouillé les droits en lecture sur ce fichier. Et si la fonction indique que le fichier est une base de données, il y a une très faible probabilité pour que ce ne soit pas le cas (vous pouvez la négliger) mais par contre il n'y a aucune assurance concernant le fait que le fichier soit intègre. Seul l'ouverture de la base de données permet de le vérifier.

CreateDB

Comme son nom l'indique, cette fonction crée une base de données. Cela consiste en fait à créer un fichier et à y stocker les informations nécessaires au fonctionnement de la base de données.

```
void CreateDB(char* DB,int nbItems);
```

Paramètres

DB : Pointeur vers une chaîne de caractère terminée par un zéro qui indique le nom du fichier dans lequel est créé la base de données.

nbItems : entier indiquant le nombre d'enregistrement que pourra au maximum contenir la base de données.

Informations

Pour l'instant le nombre d'enregistrement que contient une base de données ne peut pas être modifié, veuillez donc à ce que la valeur nbItems soit suffisamment grande.

IMPORTANT : Les enregistrements sont numérotés de 0 à (nbItems-1) lorsque vous voulez y accéder avec les autres fonctions de la bibliothèque.

Il faut aussi noter que l'appel à CreateDB n'ouvre pas la base de données, si vous voulez ensuite vous en servir, vous devrez l'ouvrir à l'aide de la fonction OpenDB.

Possibilités d'erreurs

- Une base de données est déjà ouverte, ce qui empêche la création de celle-ci.
- Le fichier dans lequel vous voulez créer la base de données existe déjà.

OpenDB

Cette fonction vous permet d'ouvrir une base de données qui a été créé auparavant. Après cela, vous pourrez lire et écrire des enregistrements existants dedans, ou en créer de nouveaux.

```
DBASE OpenDB(char* DB);
```

Paramètre

DB : Pointeur vers une chaîne de caractères terminée par un zéro qui indique le nom du fichier dans lequel se trouve la base de données que vous voulez ouvrir.

Informations

La fonction renvoie une variable de type DBASE qui identifie la base de données ouverte. Vous devrez passer ce paramètre à la plupart des fonctions qui manipulent la base de données.

Possibilités d'erreurs

- Les erreurs les plus courantes proviennent du fait que le fichier n'existe pas, ou n'est pas une base de données.

UpdateDB

Cette fonction écrit dans la base de données les informations qui ont été modifié.

```
void UpdateDB(DBASE db);
```

Paramètre

db : le descripteur de la base de données, retourné par OpenDB.

Informations

Lorsque vous écrivez un enregistrement, celui-ci est directement écrit dans le fichier de la base de données (au plus tard lorsque vous appeler CloseRecord). Par contre les modifications faites dans la structure de la table (création, destruction, ou modification de la taille des enregistrements) ne sont enregistré qu'à la fermeture de la base de données (lors de l'appel à CloseDB) cela en raison de la quantité de données à écrire. Pour des raisons de *sécurité* vous pouvez donc régulièrement appeler UpdateDB afin de forcer à la mise à jour du fichier de la base de données.

Possibilités d'erreurs

- Aucune base de données n'est ouverte.

CloseDB

Cette fonction referme la base de données actuellement ouverte.

```
void CloseDB(DBASE db);
```

Paramètre

db : le descripteur de la base de données, retourné par OpenDB.

Informations

Lors de l'appel de cette fonction, la bibliothèque appel automatiquement UpdateDB (il est donc inutile que vous le fassiez vous-même). Par contre, la bibliothèque ne ferme pas automatiquement les enregistrements ouverts. Vous devez donc absolument refermer tout les enregistrements que vous avez ouvert en lecture comme en écriture avant d'Appeler CloseDB (en réalité, vous pouvez refermer les enregistrement ouvert en mode lecture même après avoir refermé la base de données, mais je ne vous le conseille pas).

Possibilités d'erreurs

- Aucune base de données n'est ouverte.

DeleteDB

Cette fonction supprime une base de données.

```
void DeleteDB(char* DB);
```

Paramètre

DB : Pointeur vers une chaîne de caractères terminées par un zéro qui donne le nom du fichier dans lequel se trouve la base de données à supprimer.

Informations

Pour l'instant cette fonction se contente de supprimer le fichier dont le nom lui est passé en paramètre, sans vérifier s'il s'agit réellement d'une base de données. Soyez donc attentif à ce que vous faites avec cette fonction.

CompressDB

Cette fonction compacte la base de données. Ou plutôt est censé le faire : pour l'instant elle n'est pas encore programmée.

RecordExist

Indique si un enregistrement existe ou non dans la base de données.

```
bool RecordExist(DBASE db,long n);
```

Paramètres

db : le descripteur de la base de données, retourné par OpenDB.

n : entier indiquant le numéro de l'enregistrement dont on veux savoir s'il existe.

Valeur de retour

Renvoie vrai (*true*) si l'enregistrement existe et faux (*false*) sinon.

Informations

Lors de la création de la base de données, aucun enregistrement n'existe. Après cela, un enregistrement existe entre le moment où vous appelez CreateRecord avec son numéro et celui où vous appeler DeleteRecord.

CreateRecord

Crée un nouvel enregistrement dans la base de données.

```
void CreateRecord(DBASE db,long n,long mem_alloc);
```

Paramètres

db : le descripteur de la base de données, retourné par OpenDB.

n : entier indiquant le numéro de l'enregistrement a créer.

mem_alloc : entier indiquant le nombre d'octet qu'il faut allouer à l'enregistrement.

Informations

Toute la mémoire que vous demandez pour l'enregistrement peut être utilisé dans le fichier de la base de données. Donc si vous surestimer trop cette taille, votre fichier peut devenir inutilement lourd. D'un autre coté, modifié la taille d'un enregistrement est assez lourd (surtout si l'enregistrement est très grand), je ne vous conseille donc pas de modifier la taille d'un enregistrement plus d'une fois par seconde (c'est juste pour dire que cette opération est longue, mais qu'en informatique, tout va très vite).

Possibilités d'erreurs

- Le numéro de l'enregistrement que vous voulez créer est supérieur à la quantité d'enregistrement que vous avez spécifié lors de la création de la base de données.
- Un enregistrement avec ce numéro existe déjà.

DeleteRecord

Supprime un enregistrement de la base de données.

```
void DeleteRecord(DBASE db,long n);
```

Paramètres

db : le descripteur de la base de données, retourné par OpenDB.

n : entier donnant le numéro de l'enregistrement à effacer de la base de données.

Informations

Comme avec un véritable système de fichier, cette fonction ne supprime pas réellement l'enregistrement, mais supprime les références qui y sont faites dans les différentes tables de la base de données.

Possibilités d'erreurs

- L'enregistrement n'existe pas.

OpenRecord

Ouvre un enregistrement pour vous permettre de lire ou d'écrire dedans.

```
RECORD OpenRecord(DBASE db,long n,int open_mode);
```

Paramètres

db : le descripteur de la base de données, retourné par OpenDB.
n : entier donnant le numéro de l'enregistrement à ouvrir dans la base de données.
open_mode : entier donnant le mode d'ouverture de l'enregistrement.

Informations

Quatre modes d'ouverture d'un enregistrement sont possibles : MODE_READ, MODE_WRITE, MODE_APPEND et MODE_DELETE. En lecture (MODE_READ) vous ne pourrez que faire des opérations de lectures de l'enregistrement en partant du début de celui-ci. En mode écriture (MODE_WRITE) vous ne pourrez que faire des opérations d'écriture (le pointeur d'écriture se trouve au début du fichier après l'ouverture). En mode ajout (MODE_APPEND) l'enregistrement est conservé et vous pouvez écrire à partir de la fin des données actuellement enregistrées dans l'espace alloué à l'enregistrement mais que vous n'avez pas encore utilisé. En remplacement (MODE_DELETE) l'enregistrement est effacé lorsque vous l'ouvrez, mais il n'est pas supprimé de la base de données (c'est-à-dire que la mémoire que vous lui avez alloué avec CreateRecord lui est toujours alloué) et vous pouvez écrire dessus à partir du début de l'enregistrement.

La fonction retourne une variable de type RECORD, que vous devrez passez aux fonctions de manipulation des enregistrements (pour la lecture ou l'écriture principalement).

Vous pouvez ouvrir simultanément plusieurs enregistrements, ou plusieurs fois le même enregistrement. Mais si vous ouvrer plusieurs fois le même enregistrement, vous ne devez le faire qu'une fois en écriture au maximum (soit MODE_WRITE soit MODE_APPEND). La bibliothèque vous laissera les ouvrir en écriture un plus grand nombre de fois, mais le résultat est alors indéterminé et des données peuvent être perdues. De plus, si vous ouvrez en lecture et en écriture un même enregistrement. Seul les données qui existaient déjà (et qui n'ont pas été effacées) lors de l'ouverture en lecture de l'enregistrement sont accessible en lecture.

Possibilités d'erreurs

- L'enregistrement n'existe pas.

WriteRecord

Ecrit des données à la position courante dans un enregistrement ouvert en écriture.

```
void WriteRecord(RECORD record,const void* buffer,long size);
```

Paramètres

record : identificateur d'un enregistrement.
buffer : pointeur vers la zone en mémoire où se trouvent les données à écrire.
size : entier spécifiant le nombre d'octet à écrire.

Informations

L'enregistrement doit avoir été ouvert avec MODE_WRITE ou MODE_APPEND. Lorsque vous utilisez cette fonction, les données sont écrites à la position courante du pointeur d'écriture de l'enregistrement qui est ensuite déplacé à la fin des données écrites.

Possibilités d'erreurs

- L'enregistrement a été ouvert en mode lecture.
- La mémoire alloué à l'enregistrement n'est pas suffisante pour les données que vous voulez y écrire. Utilisez ResizeRecord.

ReadRecord

Lis des données à la position courantes dans un enregistrement ouvert en lecture.

```
void ReadRecord(RECORD record,void* buffer,long size);
```

Paramètres

record : identificateur d'un enregistrement.
buffer : pointeur vers la zone en mémoire où enregistrer les données à lire.
size : entier spécifiant le nombre d'octet à lire.

Informations

Lis des données à partir de la position courante du pointeur de lecture dans l'enregistrement. Après la lecture, la position du pointeur est mise à jour à la fin des données lues.

Cette fonction ne permet de lire que les données qui ont été effectivement écrites dans l'enregistrement et non pas l'intégralité de la mémoire qui a été alloué à cet enregistrement.

Possibilités d'erreurs

- L'enregistrement a été ouvert en écriture.
- Il n'y a plus assez de données à lire dans l'enregistrement.

SeekRecord

Cette fonction sers à modifier la position du pointeur de lecture ou d'écriture à l'intérieur d'un enregistrement préalablement ouvert.

```
void SeekRecord(RECORD record,long offset,int mode);
```

Paramètres

record : identificateur d'un enregistrement ouvert.
offset : L'adresse (en octet) à laquelle il faut déplacer le pointeur.
mode : entier qui spécifie relativement à où est-ce que l'offset est donné.

Informations

Le paramètre mode peut prendre deux valeurs : MODE_ABS auquel cas le pointeur est positionné à offset octets du début de l'enregistrement et MODE_REL auquel cas le pointeur est positionné à offset (qui peut être un entier négatif) octets de la position actuelle du pointeur.

Si l'enregistrement est ouvert en mode lecture, vous ne pouvez positionner le pointeur qu'à l'intérieur des données qui ont déjà été écrites dans l'enregistrement (et qui l'étaient au moment où l'enregistrement a été ouvert). Si l'enregistrement est ouvert en mode écriture, alors vous pouvez positionner le pointeur où vous voulez dans la mémoire qui a été alloué à l'enregistrement. Mais si vous le placez au-delà des données déjà écrites, alors il peu y avoir un « trou » sans données au milieu de l'enregistrement (temps que vous n'écrivez rien dedans) et la bibliothèque ne vous empêchera pas de lire dedans.

Possibilités d'erreurs

- La position demandée se situe avant le début de l'enregistrement.
- La position demandée se situe après les données écrites dans l'enregistrement (en mode lecture) ou après la fin de la zone mémoire allouée à l'enregistrement (en mode écriture).

CloseRecord

Ferme en enregistrement ouvert et mets à jour les données le concernant dans les tables de la base de données.

```
void CloseRecord(RECORD record);
```

ResizeRecord

Modifie la mémoire alloué à un enregistrement.

```
void ResizeRecord(DBASE db,long n,long mem_alloc);
```

Paramètre

db : le descripteur de la base de données, retourné par OpenDB.

GetRecordInfo

Renvoie diverses informations concernant un enregistrement ouvert.

```
record_info GetRecordInfo(RECORD record);
```

UpdateRecord

Mets à jours les informations sur un enregistrement qui a été modifié depuis son ouverture.

```
void UpdateRecord(RECORD record);
```

Paramètre

record : l'identificateur de l'enregistrement que vous voulez mettre à jour.

Informations

Cette fonction effectue deux choses différentes. Tout d'abord, comme la fonction UpdateDB, elle met à jours les informations concernant un enregistrement dans les tables de la base de données. Mais pour que ces changements soient enregistrés dans le fichier de la base de données, vous devez faire appel à UpdateDB après avoir appelé cette fonction. Toutefois mettre à jour les données concernant un enregistrement n'est utile que si vous avez écrit dans l'enregistrement au-delà des données qui y étaient précédemment. L'autre intérêt de cette fonction est de mettre à jour les données dans l'autre sens : si jamais vous avez ouvert plusieurs fois le même enregistrement (ce que je ne vous recommande pas, surtout si c'est plusieurs fois en écriture) alors cette fonction tente de récupérer les informations les plus à jours sur l'enregistrement (particulièrement si vous avez fait appel à ResizeRecord après l'ouverture de l'enregistrement) pour vous permettre de lire et d'écrire toutes les données qui se trouvent dans l'enregistrement.

Cette fonction fait à peu près la même chose que si vous fermiez l'enregistrement puis le rouvriez, mais là, la position du pointeur d'écriture (entre autre) est conservé.

GetNumberOfEmptyRecord

Renvoie le nombre d'enregistrement inutilisé dans la base de données.

```
int GetNumberOfEmptyRecord(DBASE db,);
```

Paramètre

db : le descripteur de la base de données, retourné par OpenDB.

GetFirstEmptyRecord

Renvoie le numéro du premier enregistrement inutilisé dans la base de données.

```
int GetFirstEmptyRecord(DBASE db);
```

Paramètre

db : le descripteur de la base de données, retourné par OpenDB.

GetErrorString

Renvoie le texte associé à une erreur.

```
char* GetErrorString(int error);
```

Paramètre

error : le numéro de l'erreur dont vous voulez le texte.

Informations

La fonction copie dans une zone tampon la chaîne de caractère associé à l'erreur de numéro error. La mémoire est allouée globalement, donc vous n'avez pas à vous soucier d'initialiser ou de détruire ce pointeur. Par contre, si vous appeler cette fonction dans un autre thread, la chaîne sera remplacée.

Possibilités d'erreurs

- Le numéro ne correspond à aucune erreur prévue.

Constantes

Numéro d'erreur

Les numéros d'erreur sont donnés tels quels. Certains peuvent ne correspondre à aucune erreur que le programme générera. Pour obtenir plus d'information sur une erreur particulière, appelez GetErrorString avec en paramètre le numéro de cette erreur.

ERROR_SUCCESS	0
ERROR_COULD_NOT_OPEN_FILE	1
ERROR_READ_ERROR	2
ERROR_WRITE_ERROR	3
ERROR_SEEK_ERROR	4
ERROR_LT_CORRUPTED	5
ERROR_FAT_CORRUPTED	6
ERROR_LT_DOES_NOT_EXIST	7
ERROR_FAT_DOES_NOT_EXIST	8
ERROR_RECORD_DOES_NOT_EXIST	9
ERROR_RECORD_OUT_OF_BOUND	10
ERROR_COULD_NOT_ALLOCATE_MEMORY	11
ERROR_DB_ALREADY_OPENED	12
ERROR_COULD_NOT_CREATE_FILE	13
ERROR_FILE_ALREADY_EXIST	14
ERROR_DATABASE_IS_NOT_OPEN	15
ERROR_COULD_NOT_DELETE_FILE	16
ERROR_RECORD_ALREADY_EXIST	17
ERROR_CLOSE_ERROR	18
ERROR_INVALIDE_MODE	19
ERROR_RECORD_OUT_OF_MEMORY	20
ERROR_WRONG_OPEN_MODE	21
ERROR_NO_EMPTY_RECORD	22
ERROR_SIZE_SMALLER_THAN_RECORD	23
ERROR_SEEKING_OUT_OF_RECORD	24
ERROR_UNKNOWN_ERROR	25