

ANDRE Valentin  
DEFOSSE Benjamin  
DUQUENNOY Antoine  
NOVEL Mathias  
MARCO Florian

2016/2017

## DESCRIPTIF DU CODE

La structure et la description du code se trouve dans les headers suivant :

### ***elf\_reader.h***

```
/*=====
                                STRUCTURE
=====*/

/*
Structure permettant de sauvegarder un symbole ainsi que son fichier d'origine
Utilse pour la fusion et connaitre l'origine d'un symbole
*/
typedef struct symbole {
    Elf32_Sym symbole;
    int fichier;
} Symbole;

/*=====
                                FONCTION D'EN TETE //elf_header.c
=====*/

/*
Fonction: permet de remplir la structure Elf32_Ehdr qui contiendra l'en-tête du fichier elf
@Param: FILE* f: un fichier au format elf
@Return: Elf32_Ehdr: une structure Elf32_Ehdr rempli
*/
Elf32_Ehdr lectureheader(FILE* f);

/*
Fonction: affiche l'en-tête d'un fichier elf
@Param: Elf32_Ehdr header_elf: une structure elf32_Ehdr
*/
void affichageheader(Elf32_Ehdr header_elf);

/*
Fonction: affichage du nombre magique d'un fichier elf
@Param: Elf32_Ehdr header_elf: une structure elf32_Ehdr
*/
void affichageNumMagic(Elf32_Ehdr header_elf);
```

```
/*
  Fonction: affichage de la classe et de l'endianess
  @Param: Elf32_Ehdr header_elf: une structure elf32_Ehdr
*/
void affichageMagicClasseEndianABIType(Elf32_Ehdr header_elf);

/*
  Fonction: affichage de la machine
  @Param: Elf32_Ehdr header_elf: une structure elf32_Ehdr
*/
void affichageMachine(Elf32_Ehdr header_elf);

/*
  Fonction: affichage de la partie other
  @Param: Elf32_Ehdr header_elf: une structure elf32_Ehdr
*/
void affichageOthers(Elf32_Ehdr header_elf);

/*
  Fonction: Permet de changer l'endianess de valeur selon l'endianess, sur 32 bits
  @Param: uint32_t valeur: la valeur à changer
  @Param: char endianess: l'endianess du fichier elf
  @Return: uint32_t: la valeur changé
*/
uint32_t Swap32(uint32_t valeur, char endianess);

/*
  Fonction: Permet de changer l'endianess de valeur selon l'endianess, sur 16 bits
  @Param: uint16_t valeur: la valeur à changer
  @Param: char endianess: l'endianess du fichier elf
  @Return: uint16_t: la valeur changé
*/
uint16_t Swap16(uint16_t valeur, char endianess);

/*=====
  FONCTION EN TETE DE SECTION // elf_sectionTable.c
  =====*/

/*
  Fonction: permet de remplir un tableau de structure de Elf32_Shdr à partir d'un fichier ua
  format elf
  @Param: FILE* f: le fichier elf de référence
  @Param: Elf32_Ehdr header_elf: la structure de l'en-tête du fichier elf
*/
```

```
@Param: Elf32_Shdr *t: un pointeur sur une structure Elf32_Shdr
@SideEffect: Remplissage du tableau de structure pointé par t
*/
void lectureTableSection(FILE* f, Elf32_Ehdr header_elf, Elf32_Shdr *t);

/*
Fonction: Permet d'afficher le nom d'une section
@Param: Elf32_Ehdr header: En tête du fichier elf de référence
@Param: Elf32_Shdr *section_header: un pointeur de structure Elf32_Shdr
@Param: FILE* elf: le fichier elf de référence
@Param: Elf32_Shdr headerCourant: En tête courant pour lequel on souhaite afficher
son nom
@return: int: taille du nom de la section
*/
int afficherNomSection(Elf32_Ehdr header, Elf32_Shdr *section_header, FILE* elf,
Elf32_Shdr headerCourant);

/*
Fonction: Permet d'afficher toute une table des en-têtes de section
@Param: Elf32_Shdr *section_elf: un tableau d'elf32_Shdr contenant toutes les en-têtes
de section
@Param: Elf32_Ehdr header_elf: l'en-tête du fichier elf de référence
@Param: FILE* f: le fichier elf de référence
*/
void affichageTabsection(Elf32_Shdr *section_elf, Elf32_Ehdr header_elf, FILE* elf);

/*=====
FONCTION D'AFFICHAGE DU CONTENU DES SECTIONS //
elf_recherche_section.c
=====*/

/*
Fonction: permet de retrouver une section grace à son nom
@Param: Elf32_Ehdr fileHeader: la structure d'en-tête du fichier elf de référence
@Param: Elf32_Shdr *sections_headers: Un tableau de structure Elf32_Shdr dans lequel
on va rechercher la section
@Param: FILE* elf: le fichier elf de référence
@Param: char* secName: le nom de la section à rechercher
@Return: Elf32_Shdr: une structure d'en-tête de section
*/
Elf32_Shdr getSectionByName(Elf32_Ehdr fileHeader, Elf32_Shdr *sections_headers,
FILE* elf, char * secName);

/*
Fonction: permet de retrouver une section grace à son index dans la table
@Param: Elf32_Ehdr fileHeader: la structure d'en-tête du fichier elf de référence
```

@Param: Elf32\_Shdr \*sections\_headers: Un tableau de structure Elf32\_Shdr dans lequel on va rechercher la section

@Param: File\* elf: le fichier elf de référence

@Param: int secNum: un indice dans le tableau

@Return: Elf32\_Shdr: une structure d'en-tête de section

\*/

```
Elf32_Shdr getSectionByIndex(Elf32_Ehdr fileHeader, Elf32_Shdr *sections_headers, int secNum);
```

/\*

Fonction: permet d'afficher le contenu d'une section

@Param: Elf32\_Shdr currentSection: la section dans laquelle on veut lire le contenu

@Param: FILE\* elf: le fichier elf de référence

\*/

```
void afficheContenu(Elf32_Shdr currentSection, FILE* elf);
```

/\*=====

FONCTION DE GESTION DE LA TABLE DES SYMBOLES //

elf\_symboleTable.c

=====\*/

/\*

Fonction: Permet d'afficher une table des symboles d'un fichier au format elf

@Param: Symbole\* tabSymbole: Le tableau de structure dans lequel on va lire les symboles pour les afficher

@Param: int sizeTabSymbole: la taille du tableau tabSymbole

@Param: FILE\* f: le fichier f de référence

@Param: Elf32\_Shdr\* tabSection: le tableau des en-têtes de sections

@Param: Elf32\_Ehdr header: l'en-tête du fichier elf de référence

\*/

```
void affichageTableSymbole(Symbole* tabSymbole, int sizeTabSymbole, FILE* f, Elf32_Shdr* tabSection, Elf32_Ehdr header);
```

/\*

TODO

\*/

```
void affichageTableSymboleDynamique(Elf32_Sym* tabSymboleDynamique, int sizeTabSymbole, FILE* f, Elf32_Shdr* tabSection, Elf32_Ehdr header);
```

/\*

Fonction: permet de remplir un tableau de structure de Symbole

@Param: Symbole\* tabSymbole: un tableau de Symbole contenant également une structure Elf32\_Sym qui sera rempli par effet de bord

@Param: Elf32\_Ehdr header: l'en-tête du fichier elf de référence

@Param: Elf32\_Shdr\* tabSection: Le tableau des en-têtes de section à parcourir afin de retrouver la table des symboles

@Param: FILE\* f: le fichier elf de référence

```

    @Return: int: le nombre de symbole qui aura été ajouté dans le tableau
*/
int lectureTableSymbole(Symbole* tabSymbole, Elf32_Ehdr header, Elf32_Shdr*
tabSection, FILE* f);

/*
TODO
*/
int lectureTableSymboleDynamique(Elf32_Sym* tabSymboleDynamique, Elf32_Ehdr
header, Elf32_Shdr* tabSection, FILE* f);

/*
Fonction: Permet de retrouver l'offset d'une section dans un fichier elf à partir de son nom
@param: Elf32_Ehdr header_elf: l'en tête du fichier elf de référence
@param: Elf32_Shdr *sections_table: Le tableau des en têtes de section à parcourir
@param: FILE* elf: le fichier elf de référence
@param: char * secName: Le nom de la section à retrouvé
@return: Elf32_Word: l'offset de la section secName ou -1 si non trouvé
*/
Elf32_Word rechercheOffsetSection(Elf32_Ehdr header_elf, Elf32_Shdr *sections_table,
FILE* elf, char * secName);

/*
Fonction: permet de récupérer le nom d'un symbole dans la table des chaines
@param: Elf32_Word index: offset de la table des chaines
@param: FILE* f: fichier elf de référence
@param: Elf32_Word offset: offset du nom du symbole dans la table des chaines
@param: char* c: La chaine de stockage du nom du symbole
@SideEffet: remplissage de la chaine de caractère c
*/
void recupNomSymbole(Elf32_Word index, FILE* f, Elf32_Word offset, char *c);

/*=====
                        GESTION TABLE DE REIMPLANTATION // elf_relocate.c
=====*/

/* Fonction : permet de recuperer la taille et de construire une structure de type Rela
@param : Elf32_Ehdr *file_header : Pointeur vers le header du fichier lu.
@param : Elf32_Shdr *section_headers : Pointeur vers les en-têtes de section
@param : Elf32_Rela *lesrela : Pointeur vers les tables de type Rela
@param : FILE* elf : Pointeur vers le fichier à lire
@return : La taille de l'ensemble des sections Rela.
*/

int taillerela(Elf32_Ehdr *file_header, Elf32_Shdr *section_headers, Elf32_Rela* lesrela,
FILE* elf) ;

```

/\*

Fonction : permet de recuperer la taille et de construire une structure de type Rel

@Param : Elf32\_Ehdr \*file\_header : Pointeur vers le header du fichier lu.

@Param : Elf32\_Shdr \*section\_headers : Pointeur vers les en-têtes de section

@Param : Elf32\_Rel \*lesrel : Pointeur vers les tables de type Rel

@Param : FILE\* elf : Pointeur vers le fichier à lire

@Return : La taille de l'ensemble des sections Rel.

\*/

```
int taillerel(Elf32_Ehdr *file_header, Elf32_Shdr *section_headers, Elf32_Rel* lesrel, FILE* elf);
```

/\*

Fonction : Permet d'afficher le contenu des tables de relocation dans la console

@Param : Elf32\_Ehdr \*file\_header : Pointeur vers le header du fichier lu.

@Param : Elf32\_Shdr \*section\_headers : Pointeur vers les en-têtes de section

@Param : FILE\* elf : Pointeur vers le fichier à lire

\*/

```
int affichage_relocation(Elf32_Ehdr *fileHeader, Elf32_Shdr *sections_headers, FILE* elf);
```

/\*

Fonction : Permet de compter le nombre de Table de relocation dans un fichier

@Param : Elf32\_Ehdr \*file\_header : Pointeur vers le header du fichier lu.

@Param : Elf32\_Shdr \*section\_headers : Pointeur vers les en-têtes de section

@Return : Le nombre de tables de relocations

\*/

```
int nbRel(Elf32_Ehdr *fileHeader, Elf32_Shdr *sections_headers);
```

**elf\_fusion.h :**

#include &lt;elf.h&gt;

#include &lt;stdio.h&gt;

#include &lt;byteswap.h&gt;

#include &lt;string.h&gt;

#include &lt;stdlib.h&gt;

#include &lt;unistd.h&gt;

#include &lt;getopt.h&gt;

```
/*=====
```

## STRUCTURE

```
=====*/
```

```
/*
```

Structure: Cette structure est créée pour chaque nouvelle section après la fusion

contenu: contenu brut de la section après fusion

nbOctets: taille de la section créée

Utile pour construire de manière générique le nouveau fichier objet

```
*/
```

```
typedef struct section {
```

```
    char * contenu;
```

```
    int nbOctets;
```

```
    int indexHeader;
```

```
    Elf32_Shdr header;
```

```
} Section;
```

```
/*
```

Structure: comme la structure section mais spécifique pour la section des chaînes

names: tableau à deux entrées stockant les noms

offsetCourant: offset jusqu'où l'on écrit dans le fichier

nbNames: nombre d'entrées

```
*/
```

```
typedef struct strtab {
```

```
    char ** names;
```

```
    int offsetCourant;
```

```
int nbNames;
```

```
} Strtab;
```

```
/* Structure : Super-Structure des relocalisation.
```

```
Ajout des informations concernant l'indice de la section de relocation et son nom, ainsi que  
le nombre de relocations à effectuer */
```

```
typedef struct reloctable {
```

```
    Elf32_Rel * tablerel;
```

```
    int indice_section;
```

```
    int nombre_relocation;
```

```
    char * nom_section;
```

```
} Reloactable ;
```

```
/*
```

```
    Structure: Super Structure regroupant toutes les composantes d'un fichier Elf
```

```
    Utile afin d'avoir accès à tous les éléments d'un fichier Elf
```

```
*/
```

```
typedef struct fichierElf {
```

```
    Elf32_Ehdr header_elf;
```

```
    Elf32_Shdr *sectionsTable;
```

```
    Symbole *tabSymbole;
```

```
    Elf32_Sym *tabSymboleDynamique;
```

```
    FILE *fichierElf;
```

```
    Reloactable *tabRel;
```

```
} FichierElf;
```



```
/*=====
GESTION FUSION SIMPLE ET SECTION // elf_fusionSimple.c
=====*/
```

```
/*
```

```
fonction: Permet d'initialiser le contenu de la structure de fichier Elf
```

```
@Param: FichierElf fichierELf
```

```
*/
```

```
void initFichierELF(FichierElf fichierElf);
```

```
/*
```

```
fonction: permet de libérer la mémoire allouer à la structure fichierElf passé en paramètre
```

```
@Param: FichierElf fichierELf
```

```
*/
```

```
void freeMemory(FichierElf fichierElf);
```

```
/*
```

```
fonction: permet de fusionner deux en-têtes de fichier
```

```
@Param: FichierElf *fichierElf1: la structure du premier fichier à fusionner
```

```
@Param: FichierElf *fichierElf2: la structure du deuxième fichier à fusionner
```

```
@Return: Elf32_Ehdr: une structure contenant le nouvelle en-tête du fichier Elf
```

```
*/
```

```
Elf32_Ehdr header(FichierElf *fichierElf1, FichierElf *fichierElf2);
```

```
/*
```

```
fonction: permet de recréer la section Shstrab (nom des sections)
```

@Param: Section \*section: pointeur vers la structure de section qui va contenir les éléments de Shstrab

@Param: Strtab\* shstratb: la table des chaines qui contient les éléments nécessaires pour écrire la section

@SideEffect: la section est rempli

\*/

void sectionShstrtab(Section \*section, Strtab \* shstrtab);

/\*

fonction: permet de recréer la section strab (table des chaines)

@Param: Section \*section: pointeur vers la structure de section qui va contenir les éléments de strab

@Param: Strtab\* stratb: la table des chaines qui contient les éléments nécessaires pour écrire la section

@SideEffect: la section est rempli

\*/

void sectionStrtab(Section \*section, Strtab \* strtab);

/\*

Fonction: Permet de fusionner par concatenation simple deux sections, elle va éraliser la fusion adéquate en fonction du type des deux section passer en parametre

@Param: Elf32\_Shdr sectionHeader1 : Header de section de la première section à fusionner

@Param: Elf32\_Shdr sectionHeader1 : Header de section de la seconde section à fusionner

@param: FichierElf \* fichierElf1 : la structure du premier fichier à fusionner

@param: FichierElf \* fichierElf2: la structure du second fichier à fusionner

@param: FichierElf \* fichierElfRes: la structure du fichier fusion resultat

@param: Strtab \* shstrtab : structure qui va contenir les noms de toutes les sections

@param: Strtab \* strtab : structure qui va contenir les noms de tous les symboles

@Return: Une structure Section qui possède le contenu à ecrire dans le nouveaux fichier

\*/

```
void sectionFusionSimple(Section *fusion, Elf32_Shdr sectionHeader1, Elf32_Shdr  
sectionHeader2, FichierElf * fichierElf1, FichierElf * fichierElf2);
```

/\*

Fonction: Permet de fusionner deux sections, elle va réaliser la fusion adéquate en fonction du type des deux section passer en parametre

@Param: Elf32\_Shdr sectionHeader1 : Header de section de la première section à fusionner

@Param: Elf32\_Shdr sectionHeader2 : Header de section de la seconde section à fusionner

@param: FichierElf \* fichierElf1 : la structure du premier fichier à fusionner

@param: FichierElf \* fichierElf2: la structure du second fichier à fusionner

@param: FichierElf \* fichierElfRes: la structure du fichier fusion resultat

@param: Strtab \* shstrtab : structure qui va contenir les noms de toutes les sections

@param: Strtab \* strtab : structure qui va contenir les noms de tous les symboles

@Return: Une structure Section qui possède le contenu à écrire dans le nouveaux fichier

\*/

```
Section sectionfusion(Elf32_Shdr sectionHeader1, Elf32_Shdr sectionHeader2, FichierElf  
* fichierElf1, FichierElf * fichierElf2, FichierElf * fichierElfRes, Strtab * shstrtab, Strtab *  
strtab);
```

/\*

Fonction: Permet d'ajouter une section

@Param: Elf32\_Shdr sectionHeader : Header de section de la section à ajouter

@param: FichierElf \* fichierElf : la structure du fichier contenant la fusion à ajouter

@param: Strtab \* shstrtab : structure qui va contenir les noms de toutes les sections

@Return: Une structure Section qui possède le contenu à écrire dans le nouveaux fichier

\*/

```
Section SectionAjout(Elf32_Shdr sectionHeader, FichierElf * fichierElf, Strtab * shstrtab);
```

/\*

Fonction: permet de savoir si une section est présente dans un fichierElf

@Param: FichierElf \* fichierElf: le fichier dans lequel on va rechercher la section

@Param: char \* secName: le nom de la section à rechercher

@Param: Elf32\_Shdr \* res: la section qui a été retrouvé

@Param: int \* present: un flag permettant de savoir si la section a été trouvé

\*/

```
void RechercheSectionByName(FichierElf * fichierElf, char * secName, Elf32_Shdr * res,  
int * present);
```

/\*

@Fonction: permet de récupérer le nom d'une section

// to end

\*/

```
char * getSectionName(Elf32_Shdr section, FichierElf * fichier);
```

/\*

Fonction: permet de connaître le nombre de symbole dans un fichier elf

@Param: FichierElf \*f: le fichier elf de référence

@Return: int: le nombre de symbole

\*/

```
int getNbSymbols(FichierElf *f);
```

/\*

Fonction: Permet d'écrire le nouveau fichier après la fusion

@Param: FichierElf \*fichierElfRes: la structure contenant le résultat de la fusion

@Param: Section \*sections\_elfRes: le tableau de structure de section contenant toutes les sections fusionnées

@Param: Strtab \* shstrtab: La structure contenant les noms des en têtes de section

@SideEffect: écriture dans la structure fichierElfRes du nouveau fichier elf

\*/

```
void ecritureFichierFusionnee(FichierElf *fichierElfRes, Section * sections_elfRes, Strtab * shstrtab);
```

/\*

Fonction: Fonction permettant de fusionner les deux fichiers elf et d'obtenir le résultat dans un fichier fusion

@Param: FichierElf \*fichierElf1: La structure contenant le premier fichier à fusionner

@Param: FichierElf \*fichierElf2: La structure contenant le deuxième fichier à fusionner

@Param: FichierElf \*fichierElf1: La structure contenant le résultat du fichier fusionné

@SideEffect: ecriture dans la structure contenant le fichier de la fusion

\*/

```
void fusion(FichierElf *fichierElf1, FichierElf *fichierElf2, FichierElf *fichierElfRes);
```

/\*=====

GESTION FUSION SYMBOLE // elf\_fusionSymbole.c

=====\*/

```
//Elf32_Shdr getSectionByName(Elf32_Ehdr header_elf, Elf32_Shdr *sections_table, FILE* elf, char * secName);
```

/\*

Fonction: permet de créer une nouvelle table symbole en fusionnant celles des deux fichiers d'origine

@Param: FichierElf structFicher1: structure contenant le premier fichier à fusionner

@Param: FichierElf structFicher2: structure contenant le deuxième fichier à fusionner

@Param: int sizeTab1: la taille de la table des symboles du premier fichier

@Param: int sizeTab2: la taille de la table des symboles du deuxième fichier

@Param: Symbole \*newTabSymbole: tableau de la structure qui contiendra la nouvelle table des symboles

@Param: Strtab \*strtab: pointeur vers une structure strtab

@SideEffect: écriture de la nouvelle table de symbole et mise à jour de la table des chaînes

\*/

```
int fusionTableSymbole(FichierElf structFichier1, FichierElf structFichier2, int sizeTab1, int sizeTab2, Symbole *newTabSymbole, Strtab * strtab);
```

/\*

Fonction: permet d'ajouter un nom dans la table des chaînes

@Param: char \* nom: le nom à ajouter

@Param: Strtab \*strtab: pointeur vers la structure contenant la table des chaînes

@Param: Elf32\_Sym: pointeur vers une structure de symbole

\*/

```
void AjoutNomStrtab(char * nom, Strtab * strtab, Elf32_Sym * symb);
```

/\*

Fonction: permet d'ajouter des symboles à la nouvelle table

@Param: Symbole \*newTabSymbole: Le tableau dans lequel on va ajouter les symboles

@Param: int \* nbEntree: le nombre d'entrée dans le tableau de symbole

@Param: FichierElf structFichierSrc: La structure contenant le fichier source dans lequel on va lire les symboles

@Param: int indexSrc: l'index courant vers lequel on va ajouter le symbole dans la table des symboles

@Param: char \* symbole: le symbole que l'on va ajouter

@Param: Strtab strtab: Pointeur vers la structure contenant la table des chaînes

@Param: int numFichier: le numéro identifiant le fichier source

\*/

```
void ajoutSymbole(Symbole *newTabSymbole, int * nbEntree, FichierElf structFichierSrc,  
int indexSrc, char *symbole, Strtab *strtab, int numFichier);
```

```
/*
```

Fonction: permet de créer la structure de section contenant la table des symboles après fusion

@Param: Symbole \* tableSymbole: le tableau de la structure de symbole que l'on va parcourir

@Param: int sizeTableSymbole: la taille de la table des symboles

@Param: Elf32\_Shdr structFichier1: la structure du premier fichier à fusionner

@Param: Elf32\_Shdr structFichier2: la structure du deuxième fichier à fusionner

```
*/
```

```
Section creerSectionTableSymbole(Symbole *tableSymbole, int sizeTableSymbole,  
Elf32_Shdr structFichier1, Elf32_Shdr structFichier2);
```

```
/*
```

Fonction: permet de connaître la taille d'une table section

@Param: Symbole \*tabSymbole: la table des symboles

@Param: int sizeTableSymbole: la taille de la table des symboles

@Return: une structure section

```
*/
```

```
int getSizeOfSectionTable(Symbole *tabSymbole, int sizeTableSymbole);
```

```
/*
```

Fonction: permet de connaître la valeur de ShInfo pour une section

@Param: Symbole \*tabSymbole: la table des symboles

@Param: int sizeTableSymbole: la taille de la table des symboles

@Return: la taille de la section

```
*/
```

```
int getShInfo(Symbole* tableSymbole, int sizeTableSymbole);
```

```
/*
```

Fonction: permet d'écrire l'attribut contenu d'une structure de section

@Param: Symbole \*tableSymbole: les symboles que l'on doit écrire dans le contenu

@Param: int sizeTab: la taille de la table des symboles

@Param: Section \*section: la section qui va contenir les symboles

@SideEffect: écriture du contenu dans la structure section passé en paramètre

```
*/
```

```
void EcrireContenu(Symbole *tableSymbole, int sizeTab, Section *section);
```

```
/*
```

Fonction: permet de connaître la valeur de ShIndx pour une section

@Param: Symbole symbol: un symbole

@Param: fichierElf \* fichierElf: un pointeur vers une structure fichierElf

@Param: int nbSections: le nombre de section

@Param: Strtab \*shstrtab: un pointeur vers la section shstrtab

```
*/
```

```
int getSt_shndx(Symbole symbol, FichierElf * fichierElf, int nbSections, Strtab *shstrtab);
```

```
/*=====
```

GESTION FUSION TABLE DE RELOCATION // elf\_fusionrel.c

```
=====*/
```

```
/*
```

Fonction: permet de récupérer la taille de l'ensemble des tables de relocations

@Param: FichierElf \* elfile: Un pointeur vers une structure fichierElf



@Return : La taille des tables de relocation

\*/

int relsize(FichierElf \* elfile);

/\*

Fonction : Permet de récupérer une table de Relocation et la mettre dans la structure Relactable

@Param : FichierElf \* elfile: Un pointeur vers une structure fichierElf

@Return : Une structure Relactable correctement initialisé

\*/

Relactable\* crea\_rel\_table (FichierElf \* elfile);

/\*

Fonction : Permet de fusionner deux tables de Relocation ayant le même nom de section

@Param : FichierElf \* oldelf1: Un pointeur vers une structure fichierElf

@Param : Elf32\_Shdr OldSec1 : Contenu de l'entête de section que l'on veut fusionner

@Param : Elf32\_Shdr OldSec2 : Contenu de l'entête de section que l'on veut fusionner

@Param : FichierElf \* oldelf2: Un pointeur vers une structure fichierElf

@Param : FichierElf \* newelf: Un pointeur vers une structure fichierElf

@Param : int nbnewsymbole: Un integer contenant le nombre de symbole des deux tables concantenées

@Param : int newindex : Un integer contenant le nouvel index de la section fusionnée

@Return : une structure Section correctement fusionnée

\*/

Section RelFusion(FichierElf\* oldelf1, Elf32\_Shdr OldSec1, Elf32\_Shdr OldSec2,  
FichierElf\* oldelf2, FichierElf \* newelf, int nbnewsymbole, int newindex);

/\*

Fonction : Permet de mettre à jour une table de Relocation présente uniquement dans un fichier

@Param : FichierElf \* oldelf: Un pointeur vers une structure fichierElf

@Param : FichierElf \* newelf: Un pointeur vers une structure fichierElf

@Param : Elf32\_Shdr OldSec : Contenu de l'entête de section que l'on veut mettre à jour

@Param : int nbnewsymbole: Un integer contenant le nombre de symbole des deux tables concantenées

@Return : une structure section correctement mise à jour

\*/

Section RelUpdate (FichierElf\* oldelf, FichierElf\* newelf, Elf32\_Shdr OldSec, int nbnewsymbole);

/\*

Fonction : Permet de écrire dans le fichier elf fusionne les sections.

@Param : Elf32\_Rel \*tabrel : un pointeur vers une section de relocation

@Param : int taillerel : la taille de la section de Relocation

@Param : Section \*section : Une structure de section de relocation

@Param : Elf32\_Shdr headsec : le header de la section que l'on va écrire

\*/

void EcritureStruct (Elf32\_Rel \*tabrel, int taillerel, Section \*section, Elf32\_Shdr headsec);

/\*

Fonction : Permet de initialiser les structures.

@Param : FichierElf \* oldelf1: Un pointeur vers une structure fichierElf

@Param : FichierElf \* oldelf2: Un pointeur vers une structure fichierElf

@Param : FichierElf \* newelf: Un pointeur vers une structure fichierElf

\*/

```
void init_new_rel (FichierElf* new_elf, FichierElf* oldelf1, FichierElf* oldelf2);
```