HOW-TO Programmation en C++

Al Dev (Alavoor Vasudevan) *alavoor@yahoo.com* `mailto:alavoor@yahoo.com` , traduit par Benoît Sibaud `mailto:benoit.sibaud@wanadoo.fr`

v16.0, 3 Août 2000

Ce document discute des méthodes pour éviter les problèmes de mémoire en C++ et il vous aidera aussi à programmer proprement dans ce langage. Les informations de ce document s'appliquent à tous les systèmes d'exploitation : Linux, MS-DOS, BeOS, Apple Macintosh OS, Microsoft Windows 95/98/NT/2000, OS/2, les SE d'IBM (MVS, AS/400, etc), VAX VMS, Novell Netware, tous les Unix comme Solaris, HPUX, AIX, SCO, Sinix, BSD, etc, et en résumé à tous les systèmes d'exploitation qui disposent d'un compilateur C++ (à peu près tous ceux de cette planète !).

Contents

1 Introduction

C++ est le langage le plus populaire et il sera utilisé encore longtemps dans le futur malgré l'apparition de Java. C++ s'exécute **extrêmement vite** et est en fait **10 à 20 fois plus RAPIDE** que Java. Java s'exécute très lentement car il s'agit d'un langage avec interprétation de Byte-code fonctionnant sur une machine virtuelle. Java s'exécute plus rapidement avec un compilateur à la volée (JIT - Just In Time) mais reste malgré tout plus lent que le C++. Et les programmes en C++ optimisés sont environ **3 à 4 fois plus rapides** que Java avec un compilateur à la volée ! La gestion de la mémoire en Java est automatique, donc les programmeurs n'ont pas à gérer directement les allocations de mémoire. Ce document tente d'automatiser la gestion de la mémoire en C++ pour la rendre plus facile à utiliser.

Les allocations de mémoire gérées automatiquement sont une fonctionnalité conviviale de Java. Ce document permettra au C++ d'égaler Java pour la facilité de la gestion de la mémoire.

En raison des allocations de mémoire manuelles, déboguer des programmes C++ prend une grande partie du temps de développement. Les informations de ce document vous donneront de bons trucs et astuces pour réduire le temps de débogage.

1.1 Problèmes avec les compilateurs C++ actuels

Puisque C++ est une surcouche de C, il comporte tous les défauts du C.

Par exemple, dans la programmation C, les fuites de mémoire et les débordements sont très courants en raison de l'utilisation des fonctionnalités telles que

- les types

  ```
  char *
  ```

  et

```
char[]
```

- les fonctions de manipulation de chaînes comme

```
strcpy, strcat, strncpy, strncat,
```

etc

- les fonctions de manipulation de mémoire comme

```
malloc, realloc, strdup,
```

etc

L'utilisation de **char *** et **strcpy** crée d'**affreux** problèmes dûs au *débordement (overflow)*, aux *accès hors limites (fence past errors)*, aux *"je vous écrase les orteils"* (altération des emplacements en mémoire d'autres variables) ou aux *fuites de mémoire*. Les problèmes de mémoire sont extrêmement difficiles à déboguer et très longs à corriger et à éliminer. Ils diminuent la productivité des programmeurs. Ce document aide à augmenter cette productivité grâce à différentes méthodes destinées à résoudre les défauts de la gestion de la mémoire en C++. Les bogues liés à la mémoire sont très durs à éliminer, et même les programmeurs expérimentés y passent plusieurs jours, semaines ou mois pour les déboguer. La plupart du temps, les bogues de mémoire restent "tapis" dans le code durant plusieurs mois et peuvent causer des plantages inattendus ! L'utilisation de **char *** en C++ coûte aux États-Unis et au Japon 2 milliards de dollars chaque année en temps perdu en débogage et en arrêt des programmes. Si vous utilisez **char *** en C++, cela est très coûteux, en particulier si vos programmes font plus de 50.000 lignes de code.

Par conséquent, les techniques suivantes sont proposées pour pallier les défauts du C.

Il est proposé que les compilateurs C++ devraient empêcher les programmeurs d'utiliser les types **"char *"** et **"char[]"** et les fonctions comme **strcpy**, **strcat**, **strncpy**, **strncat**. Ces types et ces fonctions sont **dangereux** et doivent être complètement **BANNIS** de la programmation en C++ ! **"char *"** est comme le *virus de la variole* et doit être éliminé de l'univers C++ ! Si vous voulez utiliser "char *", notamment avec certaines fonctions système, vous devriez utiliser le langage C. Vous mettrez alors vos programmes C dans un fichier séparé et les lierez aux programmes C++ en utilisant la directive de *spécification de lien* **extern "C"** :

```
extern "C" {
#include <stdlib.h>
}

extern "C" {
      une_fonction_c();
      une_autre_fonction_c();
}
```

La directive **extern "C"** indique que tout ce qui se trouve dans le bloc défini par les accolades (ici tout le fichier entête, `une_fonction_c()` et `une_autre_fonction_c()` sont compilés par un compilateur C).

Au lieu d'utiliser char * et char[], tous les programmeurs C++ DOIVENT utiliser la **classe String** qui est fournie dans ce document et la **classe string** inclue dans la bibliothèque standard (STDLIB). La **classe String** utilise un constructeur et un destructeur pour automatiser la gestion de la mémoire et aussi fournir de nombreuses fonctions comme *ltrim*, *substring*, etc.

Voir aussi la **classe string** du compilateur C++. La **classe string** fait partie de la bibliothèque standard GNU C++ et fournit un grand nombre de fonctions de manipulation. Les classes **string** et **String** peuvent supprimer le besoin du type **char ***. Les programmeurs C++ doivent aussi être encouragés à utiliser les opérateurs `new` et `delete` plutôt que `malloc` et `free`.

La **classe String** fait tout ce que **char *** et **char []** font. Elle peut complètement remplacer le type **char**. Il faut en plus ajouter comme avantage que les programmeurs n'auront pas du tout à s'occuper des problèmes liés à la mémoire et à son allocation !

Le compilateur GNU C++ DOIT abandonner le support des types **char * et char[]** et pour permettre la compilation de vieux programmes utilisant le type **char**, il devrait fournir une nouvelle option appelée "-fchar-datatype" pour la commande g++. Dans les deux années qui viennent, tous les programmes C++ utiliseront les **classes String et string** et il n'y aura plus ni char * ni char[]. Le compilateur devrait empêcher les mauvaises habitudes de programmation !

1.2 Que choisir entre C, C++ et Java ?

Il vous est recommandé de programmer en C++ orienté objet pour toute votre programmation généraliste ou d'applications. Vous pouvez bénéficier de tous les avantages des fonctionnalités orientées-objet du C++. Le compilateur C++ est beaucoup plus complexe que le compilateur C et les programmes en C++ peuvent s'exécuter un peu plus lentement que les programmes en C. Mais la différence de vitesse entre le C et le C++ est faible (cela peut être quelques millisecondes qui peuvent avoir un faible impact pour la programmation temps-réel). Depuis que le matériel informatique est devenu moins cher et plus rapide et la mémoire plus rapide et moins chère, il vaut mieux coder en C++ car le temps gagné grâce à la clarté et la réutilisabilité du code C++ compense la lenteur d'exécution. Les options d'optimisation comme `-O` ou `-O3` peuvent accélérer le C++/C, ce qui n'est pas possible en Java.

De nos jours, le langage C est principalement utilisé pour la programmation système pour développer les systèmes d'exploitation, les pilotes de périphériques, etc.

Note : ***En utilisant les classes String, StringBuffer, StringTokenizer et StringReader données dans ce Howto, vous pouvez coder un C++ qui ressemble "exactement" à Java ! Ce document essaie de combler le fossé entre C++ et Java, en imitant les classes Java en C++***

Java est un langage indépendant de la plateforme mieux adapté au développement d'interfaces graphiques fonctionnant dans des butineurs (Java applets, appliquettes Java) mais qui s'exécute très lentement (NdT : Java ne se limite pas aux appliquettes). Préférez l'utilisation de la programmation "Fast-CGI" en C++ du côté serveur et HTML, DHTML, XML pour obtenir de meilleures performances. À partir de là, la règle d'or est *"la programmation côté serveur en C++ et la programmation côté client (butineur) avec des appliquettes Java"*. La raison est que le système d'exploitation du côté serveur (Linux) est sous votre contrôle et ne change jamais, et que vous ne saurez jamais quel est celui du côté client/butineur. Cela peut être un terminal Internet (Linux embarqué + Netscape) ou un ordinateur avec Windows 95/98/NT/2000 ou Linux, Mac Os, OS/2, Netware, Solaris, etc.

Le gros avantage de Java est la possibilité d'exécuter des appliquettes graphiques qui fonctionnent sur n'importe quelle plateforme cliente (NdT : des applications graphiques aussi) ! Java a été créé pour remplacer les interfaces de programmation d'applications (API) de Microsoft Windows 95/NT comme MS Visual Basic ou MS Visual C++. NdT : Java a plutôt été créé pour servir de "colle universelle capable de connecter les utilisateurs aux informations" (extrait de "Au Coeur de Java" par Cay S. Horstmann et Gary Cornell). En d'autres termes, "Java est le système de fenêtrage du prochain siècle". Beaucoup de butineurs comme Netscape supportent les appliquettes et le butineur HotJava est écrit en Java. Mais le prix que vous payez pour la portabilité multi-plateformes est la baisse de performance. Les applications écrites en Java sont très lentes.

NdT : l'opposition C++/Java me semble ici réductrice. De nombreux langages de script sont utilisables et utilisés du côté serveur (Perl, Python, PHP), ainsi que les servlets et maintenant les JSP.

2 Télécharger String

Tous les programmes et exemples sont fournis dans les annexes de ce document. Vous pouvez télécharger la classe String, les bibliothèques et les exemples sous la forme d'un tar.gz :

- Allez sur `http://www.aldev.8m.com` et cliquez sur C++Programming howto.tar.gz
- Site miroir : `http://aldev.webjump.com`

3 Utilisation de la classe String

Pour utiliser la classe String, vous devez d'abord vous reporter au programme d'exemple "example_String.cpp" donné en 14 (Annexe A) et à la classe String qui se trouve en 15 (Annexe B).

La **classe String** est un remplacement complet des types `char` et `char˜*`. Vous pouvez l'utiliser exactement comme vous utilisez char et avoir encore plus de fonctionnalités. Vous devez faire l'édition des liens avec la bibliothèque 'libString.a' que vous pouvez obtenir en utilisant le Makefile fourni en 21 (Annexe H) et copier la bibliothèque dans le répertoire /usr/lib ou /lib (NdT: sous Unix) où les bibliothèques se trouvent. Pour utiliser 'libString.a', compilez vos programmes ainsi

```
g++ exemple.cpp -lString
```

Regardez l'exemple de code donné ci-dessous

```
        String aa;

        aa = " Le siège de l'ONU est à New-York ";

        // Vous pouvez utiliser aa.val() comme une variable 'char˜*' dans vos
        // programmes
        for (unsigned long tmpii = 0; tmpii < aa.length(); tmpii++)
        {
                //fprintf(stdout, "aa.val()[%ld]=%c ", tmpii, aa.val()[tmpii]);
                fprintf(stdout, "aa[%ld]=%c ", tmpii, aa[tmpii]);
        }

        // Utiliser des pointeurs sur le 'char˜*" val
        for (char˜*tmpcc = aa.val(); *tmpcc != 0; tmpcc++)
        {
                 fprintf(stdout, "aa.val()=%c ", *tmpcc);
        }
```

Un programme d'exemple complet "example_String.cpp" utilisant la classe String est donné en 14 (Annexe A) et la classe String est donnée en 15 (Annexe B).

3.1 Opérateurs

La **classe String** fournit les opérateurs suivant :

- Égal à ==
- Non égal à **!**=
- Affectation =
- Addition et affectation +=
- Concaténation des chaînes de caractères ou addition +

Exemple d'utilisation des opérateurs :

```
String aa;
String bb("Linus Torvalds")

aa = "mettez une chaîne ici";  // affectation
aa += "ajoutez-en un peu plus"; // rajout
aa = "Mon nom est" + " Alavoor Vasudevan "; // concaténation de String

if (bb == "Linus Torvalds")  // égalité booléenne
        cout << "bb est égale à 'Linus Torvalds' " << endl;

if (bb != "Alan Cox")   // non égalité booléenne
        cout << "bb n'est pas égal à 'Alan Cox'" << endl;
```

3.2 Fonctions

Les fonctions fournies par la classe String ont les **même noms** que celles de la classe String de Java. Les noms des fonctions et leur comportement sont **exactement** les mêmes que ceux de la classe String de Java ! La classe StringBuffer est aussi fournie. Cela facilitera le portage de code entre Java et C++ (vous pouvez faire du copier/coller et avoir seulement à modifier très légèrement votre code).

Le code du corps d'un fonction Java peut être copié dans le corps d'une fonction membre C++ et avec très peu de changements être compilé en C++. Un autre avantage est que les développeurs codant en Java et en C++ n'ont pas besoin de connaître deux syntaxes ou deux noms de fonctions différents.

Reportez vous en 15 (annexe B String.h) pour avoir des détails sur les noms des fonctions de la classe String.

Par exemple pour convertir un entier en chaîne, faire :

```
String  aa;

aa = 34;  // L'opérateur '=' convertira l'entier en chaîne
cout << "La valeur de aa est : " << aa.val() << endl;

aa = 234.878;  // L'opérateur '=' convertira le réel en chaîne
cout << "La valeur de aa est : " << aa.val() << endl;

aa = 34 + 234.878;
```

```
        cout << "La valeur de aa est : " << aa.val() << endl;
        // aa doit contenir '268.878' (nombre en notation anglaise)

        // transtypage
        aa = (String) 34 + " Célèbre hacker Linus Torvalds " + 234.878;
        cout << "La valeur de aa est : " << aa.val() << endl;
        // aa doit contenir '34 Célèbre hacker Linus Torvalds 234.878'
```

4 La commande C++ zap (delete)

Les commandes **delete** et **new** sont à préférer en C++ aux fonctions malloc et free du C. Prévoyez d'utiliser new et zap (commande delete) à la place de malloc et free autant que possible.

Pour rendre la commande **delete** encore plus propre, faisons une commande zap(). Définissons zap() ainsi :

```
/*
** Utilise do while pour le rendre robuste et sans erreur en cas d'utilisation
** avec les macros.
** Par exemple, si "do-while" n'est PAS utilisé alors les résultats seront
** différents comme dans
** if (bbint == 4)
**              aa = 0
** else
**              zap(aptr);  // Probleme ! aptr sera toujours mis a NULL
*/

#define zap(x) do { delete(x); x = NULL; } while (0)
```

La commande zap() libèrera la mémoire pointée et initialisera le pointeur à NULL. Cela assurera qu'en cas d'appels multiples à zap() pour un même pointeur, le programme ne plantera pas. Par exemple :

```
        zap(pFirstname);
        zap(pFirstname); // pas de plantages, car pFirstname est NULL maintenant
        zap(pFirstname); // pas de plantages, car pFirstname est NULL maintenant

        zap(pLastname);
        zap(pJobDescription);
```

Il n'y a rien de magique là-dedans, cela évite juste la répétition de code, économise le temps de frappe et rend les programmes plus lisibles. Les programmeurs C++ oublient souvent de réinitialiser à NULL les pointeurs libérés et cela cause des problèmes ennuyeux comme des 'core dumps' et des plantages. zap() gère cela automatiquement. Ne pas faire de transtypage dans la commande zap() : si quelque chose pose problème dans la commande zap() précédente, il y a probablement une autre erreur ailleurs.

De même 6 (my_malloc()), my_realloc() et my_free() devraient être utilisés à la place de malloc(), realloc() et free(), car elles sont plus propres et font des vérifications supplémentaires. Par exemple, parcourez le fichier "String.h" qui utilise les fonctions 6 (my_malloc()) et my_free().

ATTENTION : Ne pas utiliser free() pour libérer la mémoire allouée avec `new`, ni utiliser `delete` pour libérer la mémoire allouée avec malloc(). Si vous faites ça, les résultats seront imprévisibles !

5 Les pointeurs sont des problèmes

Une **référence** est un alias ; quand vous créez une référence, vous l'initialisez avec le nom d'un autre objet, la cible. À partir de ce moment, la référence est comme un autre nom de la cible, et tout ce que vous faites à la référence est vraiment fait à la cible.

Les pointeurs ne sont pas nécessaires dans la programmation générale. Dans les langages modernes comme Java, il n'y a pas de support pour des pointeurs ! Les pointeurs rendent les programmes touffus et difficiles à lire.

Si possible, évitez d'utiliser les pointeurs et utilisez les références. Les pointeurs sont vraiment très pénibles. Il est possible d'écrire une application sans utiliser de pointeurs. Dans les langages comme Java, les pointeurs n'existent pas du tout !

Syntaxe des références : déclarer une référence en indiquant le type, suivi par un opérateur référence (&), suivi par le nom de la référence. Les références **DOIVENT** être initialisées au moment de leur création. Par exemple :

```
int           poids;
int     & rpoids = poids; // NdT : prononcez ine te réf r poi

CHIEN         aa;
CHIEN & rChienRef = aa;
```

Les références : *à observer*

- Utilisez les références pour créer un alias sur un objet.
- Initialisez toutes les références.
- Utilisez les références pour les programmes nécessitant performance et efficacité.
- Utilisez **const** pour protéger les références et les pointeurs à chaque fois que cela est possible.

Les références : *à éviter*

- **IMPORTANT :** Ne pas utiliser de références sur NULL !
- Ne pas confondre l'opérateur d'adresse & avec l'opérateur référence ! Les références sont utilisées dans les sections des déclarations (voir la syntaxe des références plus haut).
- Ne pas essayer de réaffecter une référence.
- Ne pas utiliser les pointeurs si les références sont utilisables.
- Ne pas renvoyer de référence sur un objet local.
- Ne pas passer par référence si un objet référé peut sortir du champ de visibilité (scope).

6 Utilisation de my_malloc et my_free

Essayez d'éviter d'utiliser malloc et realloc si possible et utilisez **new** et **4 (zap) (delete)**. Mais parfois vous serez obligé d'utiliser les fonctions d'allocation de mémoire C en C++. Utilisez les fonctions **my_malloc()**, **my_realloc()** et **my_free()**. Ces fonctions font des allocations et des initialisations propres et essaient d'éviter les problèmes avec la mémoire. En plus ces fonctions (en mode DEBUG) peuvent garder une trace de la mémoire allouée et afficher l'usage total de la mémoire avant et après l'exécution du programme. Cela vous indique si vous avez des fuites de mémoire.

`my_malloc` et `my_realloc` sont définies ci-dessous. Elles allouent un petit peu plus de mémoire (SAFE_MEM = 5), initialisent l'espace mémoire et s'il n'est pas possible d'allouer, stoppent le programme. Les fonctions `call_check()` et `remove_ptr()` ne sont actives que si DEBUG est défini dans le Makefile et sont égales à `((void)0)` (donc NULL) pour les versions de production sans débogage. Elles permettent de suivre l'utilisation totale de la mémoire.

```
void *local_my_malloc(size_t size, char fname[], int lineno)
{
        size_t  tmpii = size + SAFE_MEM;
        void *aa = NULL;
        aa = (void *) malloc(tmpii);
        if (aa == NULL)
                raise_error_exit(MALLOC, VOID_TYPE, fname, lineno);
        memset(aa, 0, tmpii);
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

char *local_my_realloc(char *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpjj = 0;
        if (aa) // aa !=  NULL
                tmpjj = strlen(aa);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (char) * (tmpqq);
        aa = (char *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);

        // pas memset!! memset(aa, 0, tmpii);
        aa[tmpqq-1] = 0;
        unsigned long kk = tmpjj;
        if (tmpjj > tmpqq)
                kk = tmpqq;
        for ( ; kk < tmpqq; kk++)
                aa[kk] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}
```

Voir 17 (my_malloc.cpp) et le fichier d'entête 18 (my_malloc.h) pour une implantation complète de `my_malloc`.

Un exemple de l'utilisation de `my_malloc` et de `my_realloc` :

```
        char    *aa;
        int     *bb;
        float   *cc;
```

```
        aa = (char *) my_malloc(sizeof(char)* 214);
        bb = (int *) my_malloc(sizeof(int) * 10);
        cc = (float *) my_malloc(sizeof(int) * 20);

        aa = my_realloc(aa, sizeof(char) * 34);
        bb = my_realloc(bb, sizeof(int) * 14);
        cc = my_realloc(cc, sizeof(float) * 10);
```

Remarquez que dans `my_realloc` vous n'avez pas besoin de transtyper car la variable elle-même est passée et que le bon `my_realloc` qui retourne le pointeur sur le bon type est appelé. `my_realloc` est surchargée pour char*, int* et float*.

7 Les fichiers pour le débogage

Pour déboguer n'importe quel programme C++ ou C, incluez le fichier 19 (debug.h) et dans votre 'Makefile', définissez DEBUG pour activer le suivi de la mémoire dans les fonctions de debug.h. Quand vous supprimez le '-DDEBUG', les appels de la fonction de débogage sont remplacés par ((void)0) i.e. NULL, ainsi cela n'a aucun impact sur la version finale de production de votre projet. Vous pouvez généreusement utiliser les fonctions de débogage dans vos programmes et cela n'augmentera pas la taille de l'exécutable de la version de production.

Voir le fichier 20 (debug.cpp) pour l'implantation des routines de débogage. Et aussi le fichier 17 (my_malloc.cpp) comme exemple d'utilisation de debug.h et des fonctions de débogage.

Voir l'exemple de 21 (Makefile) .

8 Documentation C++ en ligne

Visitez les sites suivants sur le C++ :

- le site C++ Crash-proof `http://www.troubleshooters.com/codecorn/crashprf.htm`
- le site sur la mémoire en C++ `http://www.troubleshooters.com/codecorn/memleak.htm`

Internet contient de grandes quantités de documentation sur le C++. Visitez les moteurs de recherche comme Yahoo, Lycos, Infoseek, Excite. Taper un des mots-clés **'C++ tutoriels'**, **'C++ références'** ou **'C++ livres'**. Vous pouvez préciser les critères de recherche en cliquant sur *Recherche avancée* et en choisissant *recherche par phrase exacte*.

- `http://www.yahoo.fr`
- `http://www.lycos.fr`
- `http://www.infoseek.com` (NdT : moteur en anglais)
- `http://www.excite.com` (NdT : moteur en anglais)
- `http://www.mamma.com` (NdT : moteur en anglais)

8.1 Tutoriels C++

Il y a de nombreux tutoriels en ligne disponibles sur Internet. Tapez 'tutoriels C++' dans un moteur de recherche.

8.2 Les standards de codage C++

Visitez les URL des standards de codage C++ (NdT : sites en anglais)

- Le standard de codage C++ `http://www.cs.umd.edu/users/cml/cstyle/CppCodingStandard.html`
- Les standards de codage par Possibility `http://www.possibility.com/Cpp/CppCodingStandard.html`
- Les standards de codage par Ambysoft `http://www.ambysoft.com/javaCodingStandards.html`
- Règles et recommandations `http://www.cs.umd.edu/users/cml/cstyle/`
- Indentation et annotation `http://www.cs.umd.edu/users/cml/cstyle/indhill-annot.html`
- Règles élémentaires `http://www.cs.umd.edu/users/cml/cstyle/Ellemtel-rules.html`
- Documentation sur le style en C++ `http://www.cs.umd.edu/users/cml/cstyle/Wildfire-C++Style.html`

8.3 Référence rapide pour le C++

Tapez 'C++ Référence' dans un moteur de recherche.

8.4 Les forums de discussion Usenet sur C++

- Forum annonces C++ : `comp.lang.c++.announce`
- Forum C++ : `comp.lang.c++.*`
- (NdT : et leurs équivalents fr.comp.lang.c++.*)

9 Outils pour la mémoire

Utilisez les outils de débogage de la mémoire suivants :

- Dans le cédérom des contributions pour Linux (NdT : dans certaines distributions), voir le paquetage mem_test*.rpm
- Dans le cédérom de Linux (NdT : dans certaines distributions), voir le paquetage ElectricFence*.rpm
- Purify de Rational Software Corp `http://www.rational.com`
- Insure++ de Parasoft Corp `http://www.parasoft.com`
- Linux Tools chez `http://www.xnet.com/~blatura/linapp6.html#tools`
- Recherchez dans les moteurs de recherche comme Yahoo, Lycos, Excite, Mamma.com avec les mots-clés "Linux mémoire débogage outils".

10 URLs

Il FAUT que vous utilisiez un éditeur gérant les couleurs comme 'Vim' (Vi improved) ou Emacs lorsque vous codez en C++. Les éditeurs avec coloration syntaxique augmentent grandement votre productivité. Suivez l'URL pour le Vim HOWTO ci-dessous.

Visitez les sites suivants qui sont liés au C, au C++ :

- l'éditeur Vim avec coloration syntaxique pour le C++ et le C

  `http://metalab.unc.edu/LDP/HOWTO/Vim-HOWTO.html` (NdT : et aussi pour le SGML, pratique pour la traduction :-)
- HOWTO C++ Beautifier `http://metalab.unc.edu/LDP/HOWTO/C-C++Beautifier-HOWTO.html`
- Système de contrôle des codes source pour les programmes C++ (NdT : et tous les fichiers texte en général) (CVS HOWTO) `http://metalab.unc.edu/LDP/HOWTO/CVS-HOWTO.html`
- le site principal pour les 'petits plus' pour Linux `http://www.aldev.8m.com`
- le site miroir pour les 'petits plus' pour Linux `http://aldev.webjump.com`

11 Les autres formats pour ce document

Ce document est disponible en 11 formats différents : DVI, PostScript, LaTeX, Adobe Acrobat PDF, LyX, GNU-info, HTML, RTF (Rich Text Format), texte pur, pages de manuel Unix et SGML.

- Vous pouvez obtenir ce document HOWTO comme un fichier compressé pour les formats HTML, DVI, Postscript ou SGML sur

  `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/other-formats/`
- Le format texte simple est sur : `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO`
- Les traductions dans les autres langues comme le français, l'allemand, l'espagnol, le chinois, le japonais, etc sont sur

  `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO` . Toute aide pour la traduction est la bienvenue. (NdT : voir la section Traduction pour plus d'informations)

Ce document est écrit en utilisant un outil nommé "SGML-Tools" qui peut être obtenu sur `http://www.sgmltools.org`

Pour produire les différents formats, vous pouvez utiliser les commandes

- sgml2html -l fr -c latin C++Programming-HOWTO.sgml (pour générer les fichiers html)
- sgml2rtf -l fr -c latin C++Programming-HOWTO.sgml (pour générer le fichier RTF)
- sgml2latex -l fr -c latin C++Programming-HOWTO.sgml (pour générer le fichier LaTeX)

NdT : pour la version française, je recommande l'utilisation des options "–language=fr –charset=latin –papersize=a4".

Les documents LaTeX peuvent être convertis en fichiers PDF simplement en produisant une sortie Postscript en utilisant **sgml2latex** (et dvips) et en passant cette sortie par Acrobat **distill** (`http://www.adobe.com`) comme suit :

```
bash$ man sgml2latex
bash$ sgml2latex -l fr -c latin -p a4 --output=ps filename.sgml
bash$ distill filename.ps
bash$ man dvips
bash$ man ghostscript
bash$ man ps2pdf
bash$ ps2pdf input.ps output.pdf
bash$ acroread output.pdf &
```

Ou vous pouvez utiliser la commande Ghostscript **ps2pdf**. ps2pdf est une alternative à Adobe Acrobat Distiller, avec quasiment les mêmes fonctionnalités : il convertit les fichiers Postscript en fichiers PDF (Portable Document Format). **ps2pdf** est implanté comme un petit script (fichier batch) qui invoque Ghostcript en choisissant un "périphérique de sortie spécial" appelé **pdfwrite**. Pour pouvoir utiliser ps2pdf, le périphérique pdfwrite doit être avoir été inclu dans le Makefile lorsque Ghostscript a été compilé. Voir la documentation sur la compilation de Ghostscript pour plus de détails.

Ce document est disponible sur

- `http://sunsite.unc.edu/LDP/HOWTO/C++Programming-HOWTO.html`

Vous pouvez aussi trouver ce document sur les sites miroirs suivant

- `http://www.caldera.com/LDP/HOWTO/C++Programming-HOWTO.html`
- `http://www.WGS.com/LDP/HOWTO/C++Programming-HOWTO.html`
- `http://www.cc.gatech.edu/linux/LDP/HOWTO/C++Programming-HOWTO.html`
- `http://www.redhat.com/linux-info/ldp/HOWTO/C++Programming-HOWTO.html`
- Les autres sites miroirs près de chez vous (du point de vue de la topologie réseau) peuvent être trouvés sur

  `http://sunsite.unc.edu/LDP/hmirrors.html` . Choisissez un site et regardez le fichier /LDP/HOWTO/C++Programming-HOWTO.html. NdT : /LDP/HOWTO/translations/french/C++Programming-HOWTO.html pour la version française

Pour voir ce document au format dvi, utilisez le programme xdvi. Il est fourni avec le paquetage tetex-xdvi*.rpm dans une RedHat Linux et peut être trouvé via ControlPanel | Applications | Publishing | TeX. Pour lire un document dvi, utilisez la commande

```
xdvi -geometry 80x90 howto.dvi
man xdvi
```

Et redimensionnez la fenêtre avec la souris. Voir la page de manuel de xdvi. Pour naviguer, utilisez les flèches, Page Précédente et Page Suivante. Vous pouvez aussi utiliser les touches 'f', 'd', 'u', 'c', 'l', 'r', 'p' et 'n' pour se déplacer vers le haut, le bas, le centre, la page suivante, précédente, etc. Pour désactiver le menu expert, appuyez sur 'x'.

Vous pouvez lire le fichier PostScript en utilisant le programme 'gv' (ghostview) ou 'ghostscript'. Le programme ghostscript est dans le paquet ghostscript*.rpm et le programme gv est dans le paquet gv*.rpm dans une Redhat Linux. Ils peuvent être trouvés dans Panneau de contrôle | Applications | Graphisme. Le programme gv est plus convivial que ghostscript. Ghostscript et gv sont aussi disponibles sur les autres plateformes comme OS/2, Windows 95 et NT ; vous pouvez lire ce document même sous ces plateformes.

- Récupérez ghostscript pour Windows 95, OS/2, et pour tous les systèmes sur

  `http://www.cs.wisc.edu/~ghost`

Pour lire le document PostScript, utilisez la commande

```
gv howto.ps
ghostscript howto.ps
```

Vous pouvez lire le document HTML en utilisant Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser ou un des dix autres butineurs.

Vous pouvez lire les documents LaTeX et LyX en utilisant LyX, une interface utilisateur X-Window pour LaTeX.

12 Copyright / Droit de copie

13 Traduction

La traduction est due à Benoît Sibaud `mailto:benoit.sibaud@wanadoo.fr` . Je (NdT : Benoît Sibaud) vous encourage à participer au projet LDP et à sa traduction. Rejoignez la liste `mailto:traduc@traduc.org` et le site `http://www.traduc.org` .

14 Annexe A example_String.cpp

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//
// Auteur : Al Dev
// Utilisez la classe string ou cette classe
//
// Pour éviter les fuites de mémoire, une classe caractère qui gere les
// variables caractère.
// Préférez toujours l'utilisation de la classe string à char[] et char *.
//

// Pour compiler et tester ce programme
// (en supposant que libString.a est dans le repertoire courant)
//              g++ example_String.cpp -L. -lString

#include <stdlib.h>  // pour putenv
#include "String.h"
```

```
//#include <string>  // ceci se trouve dans /usr/include/g++-2/string
//#include <cstring> // ceci se trouve dans /usr/include/g++-2/cstring et inclut /usr/include/strings.h
//
void java_string();
void java_string_buffer();
void java_string_to_numbers();
void java_string_tokenizer();
void java_string_reader();
void java_string_writer();

//////////////////////////////////////////////////
// Un exemple de programme pour présenter String
// Note : dans cet exemple, je n'ai pas du tout
// utilisé les fonctions de manipulation de la
// memoire comme new, delete, malloc, strdup.
// La classe String gère cela automatiquement !
//////////////////////////////////////////////////

int main(int argc, char **argv)
{
        //char p_name[1024];
        //sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
        //putenv(p_name);
        //print_total_memsize(); // au debut

        java_string();
        java_string_buffer();
        java_string_to_numbers();
        java_string_tokenizer();
        java_string_reader();
        java_string_writer();

        //print_total_memsize(); // à la fin
}

// Exemple de démonstration de l'imitation de la classe
// Java String
void java_string()
{
        String aa, bb, egg;
        char  tmpaa[100];

        //bb.str_cpy("  bbSTRing  ");
        bb = "   bbSTRing   ";

        // Test de l'opérateur + à droite
        // aa + " droite "; // Aucun affichage ici !
        // Vous devez l'utiliser directement avec fprintf comme suit
        fprintf(stdout, "1) aa.val() is :%sEOF\n", (aa + " my rhs " ).val());
```

```
// Test de l'opérateur =
aa = " gauche " ;
fprintf(stdout, "2) With operator= aa.val() is :%sEOF\n", aa.val());

// Test de l'opérateur + à gauche
// " gauche " + aa; // Aucun affichage ici !
// Vous devez l'utiliser directement avec fprintf comme suit
fprintf(stdout, "3) With lsh operator+, aa.val() is :%sEOF\n", (" my lhs " + aa ).val());

// ***************** Fonctions à la Java ********************
aa = "Some Value 2345";
fprintf(stdout, "4) aa.charAt() is :%c %sEOF\n", aa.charAt(3), aa.val());

aa = "Some Value 2345";
strcpy(tmpaa, "tmpaa value");
aa.getChars(3, 8, tmpaa, 2);
fprintf(stdout, "5) aa.getChars() is : %s %sEOF\n", tmpaa, aa.val());

aa = "Some Value 2345";
fprintf(stdout, "6) aa.toCharArray() is : %sEOF\n", aa.toCharArray());

aa = "Some2345";
if (aa.equals("Some2345"))
        fprintf(stdout, "7) aa.equals() is true : %sEOF\n", aa.val());
else
        fprintf(stdout, "7) aa.equals() is false : %sEOF\n", aa.val());

aa = "testinglettercase";
egg = "TestingLetterCase";
if (aa.equalsIgnoreCase(egg))
        fprintf(stdout, "8) egg equals aa (case insensitive) aa.val is :%sEOF\n", aa.val());
else
        fprintf(stdout, "8) egg not equals aa (case insensitive) aa.val is :%sEOF\n", aa.val());

aa = "kkktestinglettercase";
egg = "abtestingLetterCase";
if (aa.regionMatches(true, 3, egg, 2, 7))
        fprintf(stdout, "9) regionMatches is true aa.val is :%sEOF\n", aa.val());
else
        fprintf(stdout, "9) regionMatches is false aa.val is :%sEOF\n", aa.val());

//aa.str_cpy(bb.val());
aa = bb + "Some Value 2345";
egg = aa.toUpperCase();
fprintf(stdout, "10) egg.val is :%sEOF\n", egg.val());

aa = bb + "Some Value 2345";
egg = aa.toLowerCase();
fprintf(stdout, "11) egg.val is :%sEOF\n", egg.val());
```

```
aa = "Some Value 2345";
egg = "Some";
if (aa.startsWith("Some"))
//if (aa.startsWith(egg))
        fprintf(stdout, "12) aa.startsWith() is true :%sEOF\n", aa.val());
else
        fprintf(stdout, "12) aa.startsWith() is false :%sEOF\n", aa.val());

aa = "Some Value 2345";
egg = " 2345";
if (aa.endsWith(" 2345"))
//if (aa.endsWith(egg))
        fprintf(stdout, "13) aa.endsWith() is true :%sEOF\n", aa.val());
else
        fprintf(stdout, "13) aa.endsWith() is false :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
egg = "caabc";
if (aa.compareTo(egg) == 0)
        fprintf(stdout, "14) aa.compareTo() is zero :%sEOF\n", aa.val());
else
if (aa.compareTo(egg) > 0)
        fprintf(stdout, "14) aa.compareTo() is greater :%sEOF\n", aa.val());
else
if (aa.compareTo(egg) < 0)
        fprintf(stdout, "14) aa.compareTo() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
strcpy(tmpaa, "aabbb Some Value 2345");
if (aa.compareTo(tmpaa) == 0)
        fprintf(stdout, "15) aa.compareTo() is zero :%sEOF\n", aa.val());
else
if (aa.compareTo(tmpaa) > 0)
        fprintf(stdout, "15) aa.compareTo() is greater :%sEOF\n", aa.val());
else
if (aa.compareTo(tmpaa) < 0)
        fprintf(stdout, "15) aa.compareTo() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
//egg = "bbb Some Value 2345";
egg = "CCaabc";  // change values to caabc, aabc
if (aa.compareToIgnoreCase(egg) == 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is zero :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(egg) > 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is greater :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(egg) < 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is less than :%sEOF\n", aa.val());
```

```
aa = "bbb Some Value 2345";
//strcpy(tmpaa, "bbb Some Value 2345");
strcpy(tmpaa, "CAABbb Some Value 2345"); // change value to caabb, aab
if (aa.compareToIgnoreCase(tmpaa) == 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is zero :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(tmpaa) > 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is greater :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(tmpaa) < 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
strcpy(tmpaa, "Some");
egg = "Value";
fprintf(stdout, "18) aa.indexOf('S') %d :%sEOF\n", aa.indexOf('S'),  aa.val());
fprintf(stdout, "18) aa.indexOf(tmpaa) %d :%sEOF\n", aa.indexOf(tmpaa),  aa.val());
fprintf(stdout, "18) aa.indexOf(egg) %d :%sEOF\n", aa.indexOf(egg),  aa.val());

aa = "bbb Some Value Some 2345";
strcpy(tmpaa, "Some");
egg = "Some";
fprintf(stdout, "19) aa.lastIndexOf('S') %d :%sEOF\n", aa.lastIndexOf('S'),  aa.val());
fprintf(stdout, "19) aa.lastIndexOf(tmpaa) %d :%sEOF\n", aa.lastIndexOf(tmpaa),  aa.val());
fprintf(stdout, "19) aa.lastIndexOf(egg) %d :%sEOF\n", aa.lastIndexOf(egg),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "20) aa.substring(5) %s :%sEOF\n",
                aa.substring(5).val(),  aa.val());

aa = "bbb Some Value Some 2345";
strcpy(tmpaa, "Some");
egg = "Some";
fprintf(stdout, "20) aa.replace('S', 'V') %s :%sEOF\n",
                aa.replace('S', 'V').val(),  aa.val());
fprintf(stdout, "20) aa.replace(Som, Vzz) %s :%sEOF\n",
                aa.replace("Som", "Vzz").val(),  aa.val());

aa = "   bbb Some Value Some 2345   ";
fprintf(stdout, "21) aa.trim() :%sEOF val() :%sEOF\n",
                aa.trim().val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "21) aa.concat() %s val :%sEOF\n",
                aa.concat("add one").val(),  aa.val());

//aa = "bbb Some Value Some 2345";
//fprintf(stdout, "21) aa.append() %s val :%sEOF\n",
//              aa.append("add append").val(),  aa.val());
```

```
aa = "bbb Some Value Some 2345";
egg = "jjjj";
fprintf(stdout, "21) aa.insert(5, egg) %s val :%sEOF\n",
                aa.insert(5, egg).val(),  aa.val());
fprintf(stdout, "21) aa.insert(5, ch) %s val :%sEOF\n",
                aa.insert(5, 'M').val(),  aa.val());

aa = "12345678";
fprintf(stdout, "46) aa.reverse()=%s aa.val is :%sEOF\n", aa.reverse().val(), aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "21) aa.deleteCharAt(4) %s val :%sEOF\n",
                aa.deleteCharAt(4).val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "22) aa.deleteStr(3,5) %s val :%sEOF\n",
                aa.deleteStr(3,5).val(),  aa.val());

// ***************** Fin des fonctions à la Java ********************

aa = "bbb Some Value Some 2345";
fprintf(stdout, "23) aa.str_tr(bomekk, BOME) %s val :%sEOF\n",
                aa.tr("bomekk", "BOME").val(),  aa.val());

aa = "bbb Some Value Some 2345";
aa = "$1,934 100%.234";
fprintf(stdout, "24) aa.compress() %s val :%sEOF\n",
                aa.compress("$,%").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "25) aa.xrange('a', 'j') %s val :%sEOF\n",
                aa.xrange('a', 'j').val(),  aa.val());
fprintf(stdout, "25) aa.xrange('1', '8') %s val :%sEOF\n",
                aa.xrange('1', '8').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "26) aa.center(15) %s val :%sEOF\n",
                aa.center(15).val(),  aa.val());
fprintf(stdout, "26) aa.center(15, '*') %s val :%sEOF\n",
                aa.center(15, '*').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "27) aa.space(3) %s val :%sEOF\n",
                aa.space(3).val(),  aa.val());

aa = "      Some Value Some 2345";
fprintf(stdout, "28) aa.left() %s val :%sEOF\n",
                aa.left().val(),  aa.val());
fprintf(stdout, "28) aa.left(18) %s val :%sEOF\n",
                aa.left(18).val(),  aa.val());
```

```
aa = "  2345    ";
fprintf(stdout, "29) aa.right():%s val :%sEOF\n",
                aa.right().val(),  aa.val());
fprintf(stdout, "29) aa.right(5):%s val :%sEOF\n",
                aa.right(5).val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "30) aa.overlay(12345678, 4, 10, *):%s val :%sEOF\n",
                aa.overlay("12345678", 4, 10, '*').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "31) aa.at(Som) %s :%sEOF\n",
                aa.at("Som").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "32) aa.before(Som) %s :%sEOF\n",
                aa.before("Skkkom").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "33) aa.after(Som) %s :%sEOF\n",
                aa.after("Som").val(),  aa.val());

aa = "   bb some value   ";
aa.ltrim(true);
fprintf(stdout, "34) aa.val is :%sEOF\n", aa.val());

aa = "   bb some value   ";
aa.rtrim(true);
fprintf(stdout, "35) aa.val() is :%sEOF\n", aa.val());

aa = "   bb some value   ";
aa.trim(true);
fprintf(stdout, "36) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = aa + " testing newlines \n\n\n\n";
aa.chopall();
fprintf(stdout, "37) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = aa + " rhs ";
fprintf(stdout, "38) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = " lhs " + aa;
fprintf(stdout, "39) aa.val() is :%sEOF\n", aa.val());

// Sample addition of numbers
//aa = (String) 9989 + "kkk" + 33 ;
```

```
aa = 9999;
fprintf(stdout, "40) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = " lhs " + aa + " rhs " + " 9989 " + " 33 ";
fprintf(stdout, "41) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = bb + "alkja " + " 99djd " ;
fprintf(stdout, "42) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = (String) "alkja " + " 99djd " ;
fprintf(stdout, "43) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa += (String) " al dev test kkk... " + " al2 slkj" + " al3333 ";
fprintf(stdout, "44) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = aa + " add aa " + aa + aa + aa + " 1111 " + " 2222 " + aa + aa + aa + " 3333 ";
fprintf(stdout, "45) aa.val() is :%sEOF\n", aa.val());

aa = "12345678";
aa.reverse(true);
fprintf(stdout, "46) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = aa + " add aa " + aa + 1111 +" "+ 2222 + " " + 3.344 + aa;
fprintf(stdout, "47) aa.val() is :%sEOF\n", aa.val());

aa.roundd(123456.0123456789012345, 13);
fprintf(stdout, "48) double aa.val() is :%sEOF\n", aa.val());

aa.roundf(123456.0123456789, 13);
fprintf(stdout, "49) float aa.val() is :%sEOF\n", aa.val());

// Test d'égalite
aa = " AA value ";
String cc(" AA value ");
if (aa == cc)
        fprintf(stdout, "50)aa=%s and cc=%s are equal!!\n", aa.val(), cc.val());
else
        fprintf(stdout, "51)aa=%s and cc=%s are NOT equal!!\n", aa.val(), cc.val());
cc = "CC";
if (aa == cc)
        fprintf(stdout, "52)aa=%s and cc=%s are equal!!\n", aa.val(), cc.val());
else
        fprintf(stdout, "53)aa=%s and cc=%s are NOT equal!!\n", aa.val(), cc.val());
if (aa == " AA value ")
```

```
                fprintf(stdout, "54)aa=%s and string are equal!!\n", aa.val());
        else
                fprintf(stdout, "55)aa=%s and string are NOT equal!!\n", aa.val());
        if (aa == " AA valuexxx ")
                fprintf(stdout, "56)aa=%s and string are equal!!\n", aa.val());
        else
                fprintf(stdout, "57)aa=%s and string are NOT equal!!\n", aa.val());

        aa = " AA bb value 12345678 ";
        fprintf(stdout, "58) aa.length() is :%ldEOF\n", aa.length());

        aa = " AA bb value 12345678 ";
        fprintf(stdout, "59) aa.repeat(BA, 4).val=%s aa.val() is :%sEOF\n",
                        aa.repeat("BA", 4).val(), aa.val());

        aa = "";
        aa = "aa";
        if (aa.isNull())
                fprintf(stdout, "60) aa.isNull() result=true%sEOF\n", aa.val());
        else
                fprintf(stdout, "60) aa.isNull() result=false%sEOF\n", aa.val());

        aa = " some value aa";
        aa.clear();
        fprintf(stdout, "61) aa.clear() %sEOF\n", aa.val());

        aa = "  abcd efg  hijk  lmno     ";
        fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                        aa.token().val(),  aa.val());
        fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                        aa.token().val(),  aa.val());
        fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                        aa.token().val(),  aa.val());
        fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                        aa.token().val(),  aa.val());
        fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                        aa.token().val(),  aa.val());

        aa = " 2345 ";
        if (aa.isInteger()) // is true
                fprintf(stdout, "63) aa is a integer val :%sEOF\n", aa.val());
        else
                fprintf(stdout, "63) aa is NOT a integer val :%sEOF\n", aa.val());

        aa = " 23.045  ";
        if (aa.isNumeric()) // is true
                fprintf(stdout, "64) aa is a numeric val :%sEOF\n", aa.val());
        else
                fprintf(stdout, "64) aa is NOT a numeric val :%sEOF\n", aa.val());
```

```
aa = " 23045  ";
fprintf(stdout, "65) aa.toInteger()=%d val :%sEOF\n",
                aa.toInteger(), aa.val());

aa = " 230.45  ";
fprintf(stdout, "66) aa.toDouble()=%f val :%sEOF\n",
                aa.toDouble(), aa.val());

aa = " testing abcdefg";
aa.chop();
fprintf(stdout, "68) aa.chop() aa.val is :%sEOF\n", aa.val());

aa = " str1 str2 string3 abcdefg joe john hardy  ";
String *strlist;
int strcount = 0;
strlist = aa.explode(strcount);
for (int ii = 0; ii <= strcount; ii++)
{
        fprintf(stdout, "69) strlist[%d] is :%sEOF\n",
                        ii, strlist[ii].val());
}

aa = " some aa ";
cout << "\n\nPlease enter a line and hit return key : ";
aa.getline();
fprintf(stdout, "70) aa.getline() is :%sEOF\n", aa.val());

aa = " some aa ";
cout << "71) Testing << operator aa is : " << aa << endl;

aa = " some aa ";
cout << "\n\n73) Testing >> operator. Enter value for aa : ";
cin >> aa;
cout << "73) Testing >> operator aa is : " << aa << endl;

// Vous pouvez utiliser aa.val() comme une variable 'char *' dans vos programmes !
aa = " str1 str2 string3 abcdefg joe john hardy  ";
fprintf(stdout, "\n\n74) Test case using aa.val() as 'char []' variable... ");
for (unsigned long tmpii = 0; tmpii < aa.length(); tmpii++)
{
        //fprintf(stdout, "aa.val()[%ld]=%c ", tmpii, aa.val()[tmpii]);
        fprintf(stdout, "aa[%ld]=%c ", tmpii, aa[tmpii]);
}
fprintf(stdout, "\n");

// Utilisation de pointeurs sur 'char *' ...
fprintf(stdout, "\n\n75) Test case using aa.val() as 'char *' pointers... ");
aa = " str1 str2 string3 abcdefg joe john hardy  ";
for (char *tmpcc = aa.val(); *tmpcc != 0; tmpcc++)
{
```

```
                fprintf(stdout, "aa.val()=%c ", *tmpcc);
        }
        fprintf(stdout, "\n");
}

// Exemple de démonstration pour la classe StringBuffer de Java
void java_string_buffer()
{
        String str1 = "ABCD EFGHI";
        cout << "\nAssigned value to str1 " << endl;
        StringBuffer aa;
        StringBuffer bb(30);
        StringBuffer cc(str1);

        cout << "\n StringBuffer aa.length() : " << aa.length() << endl;
        cout << "\n StringBuffer aa.capacity() : " << aa.capacity() << endl;
        cout << "\n StringBuffer aa.ensureCapacity(28) : " << endl;
        aa.ensureCapacity(28);
        cout << "\n StringBuffer aa.setLength(38) : " << endl;
        aa.setLength(38);

        // En Java, vous utilisez new, en C++ supprimez new
        // StringBuffer dd = new StringBuffer("some value for string buffer");
        StringBuffer dd = StringBuffer("some value for string buffer");
        cout << "\n StringBuffer dd.charAt(3) : " << dd.charAt(3) << endl;

        dd.setCharAt(3, 'K');
        cout << "\n StringBuffer setCharAt(3) : " << dd.charAt(3) << endl;

        char ee[100];
        memset(ee, 0, 100);
        strcpy(ee, "111111111111111111111111111111111111");
        dd.getChars(2, 12, ee, 3);
        cout << "\n StringBuffer getChars(2, 12, ee, 3) : " << ee << endl;

        dd.append(str1);
        cout << "\n StringBuffer append() : " << dd << endl;

        dd.append("12345678");
        cout << "\n StringBuffer append() : " << dd << endl;

        dd.append(9);
        cout << "\n StringBuffer append() : " << dd << endl;

        dd.append(7.876);
        cout << "\n StringBuffer append() : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd");
        dd.insert(4, str1);
```

```
        cout << "\n StringBuffer insert() : " << dd << endl;

        dd.reverse();
        cout << "\n StringBuffer reverse() : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd");
        dd.deleteStr(4, 9);  // Java's delete()
        cout << "\n StringBuffer deleteStr(4,9) : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd");
        dd.deleteCharAt(6);
        cout << "\n StringBuffer deleteCharAt() : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd");
        dd.replace(3, 8, str1);
        cout << "\n StringBuffer replace() : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd. A quick brown fox.");
        dd.substring(8);
        cout << "\n StringBuffer substring(8) : " << dd << endl;

        dd.setLength(1);
        dd.append(" some value for dd akjla akja kjk");
        dd.substring(8, 14);
        cout << "\n StringBuffer substring(8) : " << dd << endl;
}

// Exemple de démonstration pour les fonctions de Java parseInt,
// parseLong, floatValue et doubleValue
void java_string_to_numbers()
{
        String str1;
        int ii, jj = 40, mm = 24;
        long kk;

        str1 = "342";
        cout << "\n string str1 is : " << str1.val() << endl;
        ii = Integer.parseInt(str1);
        cout << "\n ii is : " << ii << endl;
        ii = ii + jj;  // add some value
        cout << "\n ii after adding " << jj << " is : " << ii << endl;

        str1 = "9876543210";
        cout << "\n string str1 is : " << str1.val() << endl;
        kk = Long.parseLong(str1);
        cout << "\n kk is : " << kk << endl;
```

```
        kk = kk + mm;  // add some value
        cout << "\n kk after adding " << mm << " is : " << kk << endl;

        str1 = "3.1487389876";
        cout << "\n string str1 is : " << str1.val() << endl;
        // Note : C++ n'accepte pas ceci --> Float myflt = Float.valueOf(str1);
        // Remplacement par ...
        Float myflt(str1); // Float myflt = Float.valueOf(str1);
        float nn = myflt.floatValue();
        //cout << "\n float nn is : " << nn << endl;
        fprintf(stdout, "\n double nn is : %8.20f \n", nn);
        nn = nn + mm;  // add some value
        //cout << "\n kk after adding " << mm << " is : " << nn << endl;
        fprintf(stdout, "\n kk after adding %d is : %8.20f \n", mm, nn);

        str1 = "3.1487389876";
        cout << "\n string str1 is : " << str1.val() << endl;
        // Note : C++ n'accept pas ceci --> Double mydbl = Double.valueOf(str1);
        // Remplacement par ...
        Double mydbl(str1); // Double mydbl = Double.valueOf(str1);
        double pp = mydbl.doubleValue();
        //cout << "\n double pp is : " << pp << endl;
        fprintf(stdout, "\n double pp is : %8.20f \n", pp);
        pp = pp + mm;  // add some value
        //cout << "\n kk after adding " << mm << " is : " << pp << endl;
        fprintf(stdout, "\n kk after adding %d is : %8.20f \n", mm, pp);
}

// Exemple de démonstration pour la classe Java StringTokenizer
void java_string_tokenizer()
{
        String results;

        // met une expression arithmetique dans une String et cree
        // un analyseur/tokenizer pour la string
        String mathExpr = "4*3+2/4";

        //StringTokenizer st1 = new StringTokenizer(mathExpr, "*+-", true);
        StringTokenizer st1(mathExpr, "*+-", true);

        // tant qu'il reste des tokens, les afficher
        results += "Tokens of mathExpr:\r\n";
        while (st1.hasMoreTokens())
        {
                results = results + st1.nextToken() + "\r\n";
        }
        cout << "The results : " << results << endl;

        // crée une string avec des champs délimités avec des virgules
        // et crée un analyseur/tokenizer pour elle
```

```
        String commas = "field1,field2,field3,and field4";
        //StringTokenizer st2 = new StringTokenizer(commas, ",", false);
        StringTokenizer st2(commas, ",", true);

        // tant qu'il reste des tokens, les afficher
        results = "";
        results += "Comma-delimited tokens:\r\n";
        while (st2.hasMoreTokens())
        {
                results = results + st2.nextToken() + "\r\n";
        }
        cout << "The results : " << results << endl;

        // crée une string avec des champs délimités avec des virgules
        // et crée un analyseur/tokenizer pour elle
        String colon = "f1,f2,f3,f4,f5,f6,f7,f8:f9:k1:k2:k3:k4:k5:k6:k7:k8:k9";
        //StringTokenizer st3 = new StringTokenizer(colon, ",", false);
        StringTokenizer st3(colon, ",", true);

        // tant qu'il reste des tokens, les afficher
        results = "";
        results += "Comma-delimited tokens:\r\n";
        for (int ii = 0; st3.hasMoreTokens(); ii++)
        {
                if (ii == 3)
                        results = results + st3.nextToken(":") + "\r\n";
                else
                        results = results + st3.nextToken() + "\r\n";
        }
        cout << "The results : " << results << endl;
}

// Exemple de démonstration de la classe Java StringReader
void java_string_reader()
{
        String str1;
        str1 = "some test string abcdefgh ijklk m n op  EM";
        StringReader reader(str1);

        char curChar;
        while ((curChar = reader.read()) != -1)
        {
                cout << "curChar : " << curChar << endl;
        }
}

// Exemple de démonstration de la classe Java StringWriter
void java_string_writer()
{
        String str1;
```

```
        str1 = "some str";
        StringWriter writer;
        writer.write("Hello ");
        writer.write("xxxWorldxxx", 3, 8);
        str1 = writer.toString();
        cout << "str1 using toString() : " << str1 << endl;

        writer.close();
        str1 = writer.toString();
        cout << "str1 after doing close() : " << str1 << endl;

        str1 = "some str";
        writer.write(" Yet another Hello ");
        writer.write("xxxxxWorld-widexxx", 5, 15);
        str1 = writer.getBuffer();
        cout << "str1 using the getBuffer() : " << str1 << endl;
}
```

15 Annexe B String.h

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//
// Auteur : Al Dev
// Utilisez la classe string ou cette classe
//
// Pour prévenir les fuites de mémoire - une classe caractère qui gère les
// variables caractère.
// Préférez toujours l'utilisation de la classe String à char[] et char *.
//

#ifndef __STRING_H_
#define __STRING_H_

//#include <iostream> // ne pas utiliser iostream car le programme devient volumineux.
//#include <stdlib.h> // pour free() et malloc()
#include <string.h> // pour strcpy()
#include <ctype.h> // pour isspace()
#include <stdio.h> // pour sprintf()
#include <list.h> // pour sprintf()
#include <math.h> // pour modf(), rint()

#include "my_malloc.h"
#include "debug.h" // debug_(name, value)  debug2_(name, value, LOG_YES)

const short INITIAL_SIZE =      50;
const short NUMBER_LENGTH = 70;
const int MAX_ISTREAM_SIZE = 2048;
```

```
class StringBuffer;

// une petite classe avec le STRICT MINIMUM de fonctions et de variables
// Cette classe doit être gardée petite...
class String
{
        public:
                String();
                String(char bb[]);  // nécessaire pour operator+
                String(char bb[], int start, int slength); // sous-ensemble de caractères
                String(int bb);  // nécessaire pour operator+
                String(unsigned long bb);  // nécessaire pour operator+
                String(long bb);  // nécessaire pour operator+
                String(float bb);  // nécessaire pour operator+
                String(double bb);  // nécessaire pour operator+
                String(const String & rhs);  // constructeur de recopie nécessaire pour operator+
                String(StringBuffer sb);  // compatibilité Java
                String(int bb, bool dummy);  // pour la classe StringBuffer
                ~String();

                char *val() {return sval;} // Pas sur de mettre sval publique

                // Les fonctions ci-dessous imitent le String de Java
                unsigned long length() { return strlen(sval); }
                char charAt(int where);
                void getChars(int sourceStart, int sourceEnd,
                                char target[], int targetStart);
                char* toCharArray();
                char* getBytes();

                bool equals(String str2); // Voir aussi operator ==
                bool equals(char *str2); // Voir aussi operator ==
                bool equalsIgnoreCase(String str2);

                bool regionMatches(int startIndex, String str2,
                                int str2StartIndex, int numChars);
                bool regionMatches(bool ignoreCase, int startIndex,
                                String str2, int str2StartIndex, int numChars);

                String toUpperCase();
                String toLowerCase();

                bool startsWith(String str2);
                bool startsWith(char *str2);

                bool endsWith(String str2);
                bool endsWith(char *str2);

                int compareTo(String str2);
```

```
int compareTo(char *str2);
int compareToIgnoreCase(String str2);
int compareToIgnoreCase(char *str2);

int indexOf(char ch, int startIndex = 0);
int indexOf(char *str2, int startIndex = 0);
int indexOf(String str2, int startIndex = 0);

int lastIndexOf(char ch, int startIndex = 0);
int lastIndexOf(char *str2, int startIndex = 0);
int lastIndexOf(String str2, int startIndex = 0);

String substring(int startIndex, int endIndex = 0);
String replace(char original, char replacement);
String replace(char *original, char *replacement);

String trim(); // Voir aussi trim() surcharge

String concat(String str2);  // Voir aussi operator +
String concat(char *str2); // Voir aussi operator +

String reverse(); // Voir aussi reverse() surcharge
String deleteCharAt(int loc);
String deleteStr(int startIndex, int endIndex); // le "delete()" de Java

String valueOf(char ch)
        {char aa[2]; aa[0]=ch; aa[1]=0; return String(aa);}
String valueOf(char chars[]){ return String(chars);}
String valueOf(char chars[], int startIndex, int numChars);
String valueOf(bool tf)
        {if (tf) return String("true"); else return String("false");}
String valueOf(int num){ return String(num);}
String valueOf(long num){ return String(num);}
String valueOf(float num) {return String(num);}
String valueOf(double num) {return String(num);}

// Voir aussi la classe StringBuffer plus bas

// ---- Fin des fonctions à la Java -----

//////////////////////////////////////////////////////////////
// Liste des fonctions aditionnelles non présentes dans Java
//////////////////////////////////////////////////////////////
String ltrim();
void ltrim(bool dummy); // dummy pour avoir une signature differente
String rtrim();
void rtrim(bool dummy); // dummy pour avoir une signature differente

void chopall(char ch='\n'); // supprime les caractères 'ch' en fin
void chop(); // supprime un caractère en fin
```

```
void roundf(float input_val, short precision);
void decompose_float(long *integral, long *fraction);

void roundd(double input_val, short precision);
void decompose_double(long *integral, long *fraction);

void explode(char *seperator); // voir aussi token() et explode() surcharge
String *explode(int & strcount, char seperator = ' '); // voir aussi token()
void implode(char *glue);
void join(char *glue);
String repeat(char *input, unsigned int multiplier);
String tr(char *from, char *to); // translate characters
String center(int padlength, char padchar = ' ');
String space(int number = 0, char padchar = ' ');
String xrange(char start, char end);
String compress(char *list = " ");
String left(int slength = 0, char padchar = ' ');
String right(int slength = 0, char padchar = ' ');
String overlay(char *newstr, int start = 0, int slength = 0, char padchar = ' ');

String at(char *regx); // renvoie la première occurrence de regx
String before(char *regx); // renvoie la chaîne avant regx
String after(char *regx); // renvoie la chaîne apres regx
String mid(int startIndex = 0, int length = 0);

bool isNull();
bool isInteger();
bool isInteger(int pos);
bool isNumeric();
bool isNumeric(int pos);
bool isEmpty();  // idem que length() == 0
bool isUpperCase();
bool isUpperCase(int pos);
bool isLowerCase();
bool isLowerCase(int pos);
bool isWhiteSpace();
bool isWhiteSpace(int pos);
bool isBlackSpace();
bool isBlackSpace(int pos);
bool isAlpha();
bool isAlpha(int pos);
bool isAlphaNumeric();
bool isAlphaNumeric(int pos);
bool isPunct();
bool isPunct(int pos);
bool isPrintable();
bool isPrintable(int pos);
bool isHexDigit();
bool isHexDigit(int pos);
```

```
bool isCntrl();
bool isCntrl(int pos);
bool isGraph();
bool isGraph(int pos);

void clear();
int toInteger();
long parseLong();

double toDouble();
String token(char separator = ' '); // voir StringTokenizer, explode()
String crypt(char *original, char *salt);
String getline(FILE *infp = stdin); // voir aussi putline()
//String getline(fstream *infp = stdin); // voir aussi putline()

void putline(FILE *outfp = stdout); // voir aussi getline()
//void putline(fstream *outfp = stdout); // voir aussi getline()

void swap(String aa, String bb); // permute aa et bb
String *sort(String aa[]);  // trie le tableau de chaînes
String sort(int startIndex = 0, int length = 0);  // trie les caractères de la chaîne
int freq(char ch); // retourne le nombre d'occurrences distinctes, non superposees
void Format(const char *fmt, ...);
String replace (int startIndex, int endIndex, String str);

void substring(int startIndex, int endIndex, bool dummy);
void reverse(bool dummy); // dummy pour avoir une signature differente
String deleteCharAt(int loc, bool dummy);
String deleteStr(int startIndex, int endIndex, bool dummy);
void trim(bool dummy); // dummy pour avoir une signature differente
String insert(int index, String str2);
String insert(int index, String str2, bool dummy);
String insert(int index, char ch);
String insert(int index, char ch, bool dummy);
String insert(char *newstr, int start = 0, int length = 0, char padchar = ' ');

// requis par le StringBuffer de Java
void ensureCapacity(int capacity);
void setLength(int len);
void setCharAt(int where, char ch);

// requis par les classes Integer Long et Double de Java
int parseInt(String ss) {return ss.toInteger();}
int parseInt(char *ss)
        {String tmpstr(ss); return tmpstr.toInteger();}
long parseLong(String ss) {return ss.parseLong();}
long parseLong(char *ss)
        {String tmpstr(ss); return tmpstr.parseLong();}
float floatValue() {return (float) toDouble(); }
double doubleValue() {return toDouble(); }
```

```
//////////////////////////////////////////////////
//              Liste des fonctions dupliquées
//////////////////////////////////////////////////
// char * c_str() // utilisez val()
// bool find();  // utilisez regionMatches()
// bool search();  // utilisez regionMatches()
// bool matches(); // utilisez regionMatches()
// int rindex(String str2, int startIndex = 0); utilisez lastIndexOf()
// String blanks(int slength);  // utilisez repeat()
// String append(String str2); // utilisez concat() ou operator+
// String prepend(String str2);  // utilisez operator+. Voir aussi append()
// String split(char seperator = ' ');  // utilisez token()
bool contains(char *str2, int startIndex = 0); // utilisez indexOf()
// void empty(); utilisez is_empty()
// void vacuum(); utilisez clear()
// void erase(); utilisez clear()
// void zero(); utilisez clear()
// bool is_float(); utilisez is_numeric();
// bool is_decimal(); utilisez is_numeric();
// bool is_Digit(); utilisez is_numeric();
// float float_value(); utilisez toDouble();
// float tofloat(); utilisez toDouble();
// double double_value(); utilisez toDouble();
// double numeric_value(); utilisez toDouble();
// int int_value(); utilisez toInteger()
// int tonumber(); utilisez toInteger()
// String get(); utilisez substring() ou val() mais preferrez le substring de Java
// String getFrom(); utilisez substring() ou val() mais preferrez le substring de Java
// String head(int len); utilisez substring(0, len)
// String tail(int len); utilisez substring(length()-len, length())
// String cut(); utilisez deleteCharAt() ou deleteStr()
// String cutFrom(); utilisez deleteCharAt() ou deleteStr()
// String paste(); utilisez insert()
// String fill(); utilisez replace()
// char firstChar(); // utilisez substring(0, 1);
// char lastChar(); // utilisez substring(length()-1, length());
// String findNext(); utilisez token()

// begin();  iterator. utilisez operator [ii]
// end();  iterator. utilisez operator [ii]
// copy();  utilisez l'opérateur d'affectation, String aa = bb;
// clone();  utilisez l'opérateur d'affectation, String aa = bb;

// Tous les opérateurs ...
String operator+ (const String & rhs);
friend String operator+ (const String & lhs, const String & rhs);

String& operator+= (const String & rhs); // utiliser une référence va plus vite
String& operator= (const String & rhs); // utiliser une référence va plus vite
```

```
        bool operator== (const String & rhs); // utiliser une référence va plus vite
        bool operator== (const char *rhs);
        bool operator!= (const String & rhs);
        bool operator!= (const char *rhs);
        char operator [] (unsigned long Index) const;
        char& operator [] (unsigned long Index);
        friend ostream &  operator<< (ostream & Out, const String & str2);
        friend istream &  operator>> (istream & In, String & str2);

        static  list<String>            explodeH;  // tete de liste

    protected:
        char *sval; // Pas sûr de mettre sval publique
        inline void verifyIndex(unsigned long index) const;
        inline void verifyIndex(unsigned long index, char *aa) const;

        void _str_cat(char bb[]);
        void _str_cat(int bb);
        void _str_cat(unsigned long bb);
        void _str_cat(float bb);

    private:
        // Note : toutes les fonctions et variables privées ont des noms
        // commencant par _ (souligne)

        //static String *_global_String; // utilisé dans l'opérateur ajout
        //inline void _free_glob(String **aa);
        void _str_cpy(char bb[]);
        void _str_cpy(int bb); // itoa
        void _str_cpy(unsigned long bb);
        void _str_cpy(float bb); // itof

        bool _equalto(const String & rhs, bool type = false);
        bool _equalto(const char *rhs, bool type = false);
        String *_pString;  // pointeur temporaire pour usage interne
        inline void _allocpString();
        inline void _reverse();
        inline void _deleteCharAt(int loc);
        inline void _deleteStr(int startIndex, int endIndex);
        inline void _trim();
        inline void _ltrim();
        inline void _rtrim();
        inline void _substring(int startIndex, int endIndex);
};

// Imite StringBuffer de Java
// Cette classe est prévue pour que le code Java soit portable en C++ en ne
// nécessitant que peu de changements
// Note : si vous codez en C++, N'utilisez PAS cette classe StringBuffer,
// elle n'est fournie que pour compiler du code écrit en Java copié/collé
```

```
// dans du code C++.
class StringBuffer: public String
{
        public:
                StringBuffer();
                StringBuffer(int size);
                StringBuffer(String str);
                ~StringBuffer();

                int capacity() {return strlen(sval);}
                StringBuffer append(String str2)
                        { *this += str2; return *this;} // Voir aussi l'opérateur +
                StringBuffer append(char *str2)
                        { *this += str2; return *this;} // Voir aussi l'opérateur +
                StringBuffer append(int bb)
                        { *this += bb; return *this;} // Voir aussi l'opérateur +
                StringBuffer append(unsigned long bb)
                        { *this += bb; return *this;} // Voir aussi l'opérateur +
                StringBuffer append(float bb)
                        { *this += bb; return *this;} // Voir aussi l'opérateur +
                StringBuffer append(double bb)
                        { *this += bb; return *this;} // Voir aussi l'opérateur +

                StringBuffer insert(int index, String str2)
                        { return String::insert(index, str2, true);}

                StringBuffer insert(int index, char ch)
                        { return String::insert(index, ch, true);}

                StringBuffer reverse()
                        { String::reverse(true); return *this;}

                // le "delete()"de Java. On ne peut pas utiliser le nom delete en C++
                StringBuffer deleteStr(int startIndex, int endIndex)
                        { String::deleteStr(startIndex, endIndex, true); return *this;}
                StringBuffer deleteCharAt(int loc)
                        { String::deleteCharAt(loc, true); return *this;}

                StringBuffer substring(int startIndex, int endIndex = 0)
                        { String::substring(startIndex, endIndex, true); return *this;}
};

static String Integer("0"); // Integer.parseInt(String) de Java
static String Long("0"); // Long.parseLong(String) de Java

// Imite la classe Float et Float.floatValue() de Java
// Est fournie pour compiler du code Java en C++.
class Float: public String
{
        public:
```

```
                Float(String str);
                Float valueOf(String str2) {return Float(str2);}
                float floatValue() {return (float) toDouble(); }
};

// Imite la classe Double de Java et Double.doubleValue()
// Est fournie pour compiler du code Java en C++.
class Double: public String
{
        public:
                Double(String str);
                Double valueOf(String str2) {return Double(str2);}
                double doubleValue() {return toDouble(); }
};

// Imite la classe StringTokenizer de Java
// Fournie pour compiler du code Java en C++ et vice-versa
class StringTokenizer: public String
{
        public:
                StringTokenizer(String str);
                StringTokenizer(String str, String delimiters);
                StringTokenizer(String str, String delimiters, bool dflag);
                ~StringTokenizer();

                int     countTokens();
                bool    hasMoreElements();
                bool    hasMoreTokens();
                String  nextElement(); // en Java retourne un 'Object'
                String  nextToken();
                String  nextToken(String delimiters);
        private:
                int             _current_position; // position courante dans la chaîne
                int             _total_tokens;
                int             _remaining_tokens;
                char *  _listofdl; // liste de delimiteurs
                char *  _workstr; // chaîne temporaire
                char *  _origstr; // chaîne originalement passée
                bool    _dflag;  // indicateur de delimiteur
                inline  void _prepworkstr(char *delimiters = NULL);
};

// Imite la classe StringReader de Java
// Fournie pour compiler du code Java en C++
class StringReader: public String
{
        public:
                StringReader(String str);
                void close() {}  // ferme le flux
                void mark(int readAheadLimit);
```

```
                bool markSupported() {return true;} // indique si le flux supporte l'opération mark(), il le fait
                int read();
                int read(char cbuf[], int offset, int length);
                bool ready() {return true;} // indique si le flux est prêt à être lu
                void reset();
                long skip(long ii);
        private:
                unsigned long   _curpos;
                unsigned long   _mark_pos;
};

// Imite la classe StringWriter de Java
// fournie pour compiler du code Java en C++
class StringWriter: public String
{
        public:
                StringWriter();
                StringWriter(int bufferSize);
                void close() {clear();}
                void flush() {clear();}
                StringBuffer getBuffer() {return (StringBuffer) *this;}
                String  toString() {return (String) *this;}
                void write(int);
                void write(String);
                void write(char *str1);
                void write(char str1[], int startIndex, int endIndex);
                void write(String str1, int startIndex, int endIndex);
};

// Les variables globales sont définies dans String.cpp

#endif // __STRING_H_
```

16 Annexe C String.cpp

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//
// Auteur : Al Dev
// Utilisez la classe string ou cette classe
//
// Pour prévenir les fuites de mémoire - une classe caractère qui gère les
// variables caractères.
// Préférez toujours l'utilisation de la classe string à char[] et char *.
//
//

// Pour compiler et tester ce programme, utilisez -
```

```
//              g++ String.cpp

#include "String.h"

//#include <sys/va_list.h> pour Format()
//#include <sys/varargs.h> pour Format()

// Variables globales....
//String *String::_global_String = NULL; // variable globale
list<String>            String::explodeH;

String::String()
{
        debug_("In cstr()", "ok");
        sval = (char *) my_malloc(sizeof(char)* INITIAL_SIZE);

        _pString = NULL;
}

String::String(char *bb)
{
        unsigned long tmpii = strlen(bb);
        sval = (char *) my_malloc(sizeof(char)* tmpii);
        strncpy(sval, bb, tmpii);
        sval[tmpii] = '\0';

        //debug_("In cstr(char *bb) bb", bb);
        debug_("In cstr(char *bb) sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this-pointer=%x\n", this);
        #endif // DEBUG

        _pString = NULL;
}

String::String(char *bb, int start, int slength)
{
        unsigned long tmpii = strlen(bb);
        if (start > (int) tmpii || start < 0)
        {
                cerr << "\nString(char *, int, int) - start is  out of bounds!!\n" << endl;
                exit(1);
        }
        sval = (char *) my_malloc(sizeof(char)* slength);
        strncpy(sval, & bb[start], slength);
        sval[slength] = '\0';

        //debug_("In cstr(char *bb) bb", bb);
        debug_("In cstr(char *bb) sval", sval);
```

```
	#ifdef DEBUG
		//fprintf(stderr, "\nAddress of sval=%x\n", & sval);
		//fprintf(stderr, "\nAddress of this-pointer=%x\n", this);
	#endif // DEBUG

	_pString = NULL;
}

String::String(int bb)
{
	sval = (char *) my_malloc(NUMBER_LENGTH); // int avec 70 chiffres max
	sprintf(sval, "%d", bb);
	debug_("In cstr(int bb) sval", sval);

	_pString = NULL;
}

String::String(unsigned long bb)
{
	sval = (char *) my_malloc(NUMBER_LENGTH); // unsigned long avec 70 chiffres max
	sprintf(sval, "%lu", bb);
	debug_("In cstr(unsigned long bb) sval", sval);

	_pString = NULL;
}

String::String(long bb)
{
	sval = (char *) my_malloc(NUMBER_LENGTH); // long avec 70 chiffres max
	sprintf(sval, "%ld", bb);
	debug_("In cstr(long bb) sval", sval);

	_pString = NULL;
}

String::String(float bb)
{
	sval = (char *) my_malloc(NUMBER_LENGTH); // float avec 70 chiffres max
	sprintf(sval, "%f", bb);
	debug_("In cstr(float bb) sval", sval);

	_pString = NULL;
}

String::String(double bb)
{
	sval = (char *) my_malloc(NUMBER_LENGTH); // double avec 70 chiffres max
	sprintf(sval, "%f", bb);
	debug_("In cstr(double bb) sval", sval);
```

```
        _pString = NULL;
}

// Constructeur par recopie utilisé par l'opérateur +
String::String(const String & rhs)
{
        // Effectue une copie en profondeur à la place de la copie superficielle par défaut du compilateur
        debug_("In copy-cstr()", "ok");
        unsigned long tmpii = strlen(rhs.sval);
        sval = (char *) my_malloc(sizeof(char)* tmpii);
        strncpy(sval, rhs.sval, tmpii);
        sval[tmpii] = '\0';

        _pString = NULL;
}

// Cette fonction fournit une compatibilité avec le code Java
String::String(StringBuffer sb)
{
        debug_("In String(StringBuffer)", "ok");
        unsigned long tmpii = strlen(sb.sval);
        sval = (char *) my_malloc(sizeof(char)* tmpii);
        strncpy(sval, sb.sval, tmpii);
        sval[tmpii] = '\0';

        _pString = NULL;
}

// Utilisé par la classe StringBuffer. Utilise la variable dummy
// pour différentes signatures.
// La classe StringBuffer imite le StringBuffer de Java
String::String(int size, bool dummy)
{
        sval = (char *) my_malloc(sizeof(char)* size);
        debug_("In cstr(int size, bool dummy) sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this-pointer=%x\n", this);
        #endif // DEBUG

        _pString = NULL;
}

String::~String()
{
        debug_("In dstr sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this-pointer=%x\n", this);
        #endif // DEBUG
```

```
        my_free(sval);
        //delete [] sval;
        sval = NULL;

        delete _pString; _pString = NULL;
}

inline void String::_allocpString()
{
        // _pString will be deleted in destructor
        if (!_pString)  // if (_pString == NULL)
                _pString = new String(this->sval);
        else
                *_pString = this->sval;
}

// DOIT utiliser un pointeur-sur-pointeur **aa, sinon la mémoire utilisée
// par l'argument N'est PAS libérée !
/*
inline void String::_free_glob(String **aa)
{
        debug_("called _free_glob()", "ok" );
        if (*aa != NULL)  // (*aa != NULL)
        {
                debug_("*aa is not null", "ok");
                delete *aa;
                *aa = NULL;
        }
        //else
                debug_("*aa is null", "ok");

        //if (*aa == NULL)
        debug_("*aa set to null", "ok");
}
*/

// Imite le charAt de java.lang.String
char String::charAt(int where)
{
        verifyIndex(where);
        return (sval[where]);
}

// Imite la fonction getChars de java.lang.String
// sourceStart spécifie l'indice du début de la sous-chaîne
// et sourceEnd spécifie l'indice juste après la fin de la sous-chaîne désirée
// Ainsi la sous-chaîne contient les caractères de sourceStart jusqu'à
// (sourceEnd - 1). Le tableau qui va recevoir les caractères est target.
// targetStart est l'indice dans target à partir duquel la copie sera effectuée.
// Il convient de s'assurer que le tableau target est assez grand pour pouvoir contenir
```

```
// le nombre de caractères désiré.
// Par exemple  getChars(3, 6, aa, 0) sur "ABCDEFGHIJK" donne aa ="DEF"
void String::getChars(int sourceStart, int sourceEnd, char target[], int targetStart)
{
        verifyIndex(sourceStart);
        verifyIndex(sourceEnd);

        if (sourceEnd >= sourceStart)
        {
                strncpy(& target[targetStart], & sval[sourceStart], sourceEnd - sourceStart);
                target[targetStart + (sourceEnd - sourceStart)] = 0;
        }
        else
        {
                cerr << "\ngetChars() - SourceEnd is greater than SourceStart!!\n" << endl;
                exit(1);
        }
}

// Imite getChars de java.lang.String
// Retourne un tableau caractères contenant la chaîne entière
char* String::toCharArray()
{
        return (sval);
}

// Imite getBytes de java.lang.String
// Retourne un tableau caractères contenant la chaîne entière
char* String::getBytes()
{
        return (sval);
}

// Imite equals de java.lang.String
bool String::equals(String str2)  // voir aussi l'opérateur ==
{
        return ( _equalto(str2.sval));
}

// Imite equals de java.lang.String
bool String::equals(char *str2)  // voir aussi l'opérateur ==
{
        return ( _equalto(str2));
}

// Imite equalsIgnoreCase de java.lang.String
bool String::equalsIgnoreCase(String str2)
{
        String  aa, bb;
        aa = this->toLowerCase();
```

```
        bb = str2.toLowerCase();
        return ( aa._equalto(bb.sval) );
}

// Imite regionMatches de java.lang.String
// startIndex spécifie l'indice à partir duquel débute la région dans l'objet
// String invoquant la méthode. La chaîne est comparée à str2. L'indice à partir
// duquel la comparaison commencera dans str2 est spécifié par str2Index
// La longueur de la sous-chaîne comparée est numChars.
bool String::regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)
{
        verifyIndex(startIndex);
        str2.verifyIndex(str2StartIndex);
        if (strncmp(& this->sval[startIndex], & str2.sval[str2StartIndex], numChars) == 0)
                return true;
        else
                return false;
}

// Imite regionMatches de java.lang.String
// Il s'agit de la version surchargée de regionMatches
// Si ignoreCase vaut true, la casse des caractères est ignorée, sinon
// la casse est significative (i.e. si ignoreCase vaut true alors ignore la
// casse et compare)
// startIndex spécifie l'indice à partir duquel débute la region dans l'objet
// String invoquant la méthode. La chaîne est comparée à str2. L'indice à partir
// duquel la comparaison commencera dans str2 est spécifié par str2Index
// La longueur de la sous-chaîne comparée est numChars.
bool String::regionMatches(bool ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)
{
        if (ignoreCase)  // if (ignoreCase == true)
        {
                verifyIndex(startIndex);
                str2.verifyIndex(str2StartIndex);
                String string1, string2;
                string1 = this->toLowerCase();
                string2 = str2.toLowerCase();
                if (strncmp(& string1.sval[startIndex], & string2.sval[str2StartIndex], numChars) == 0)
                        return true;
                else
                        return false;
        }
        else
        {
                return regionMatches(startIndex, str2, str2StartIndex, numChars);
        }
}

// Imite toLowerCase de java.lang.String
//       String  ss("sometest");
```

```
//       String  egg = ss.toLowerCase();
String String::toLowerCase()
{
        _allocpString();

        for (long tmpii = strlen(_pString->sval); tmpii >= 0; tmpii--)
        {
                _pString->sval[tmpii] = tolower(_pString->sval[tmpii]);
        }
        return *_pString;  // return the object now
}

// Imite toUpperCase de java.lang.String
//       String  ss("sometest");
//       String  egg = ss.toUpperCase();
String String::toUpperCase()
{
        _allocpString();

        for (long tmpii = strlen(_pString->sval); tmpii >= 0; tmpii--)
        {
                _pString->sval[tmpii] = toupper(_pString->sval[tmpii]);
        }
        return *_pString;  // return the object now
}

// Imite startsWith de java.lang.String
bool String::startsWith(String str2)
{
        if (!strncmp(this->sval, str2.sval, strlen(str2.sval) )) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imite startsWith de java.lang.String
// Fonction surchargée
bool String::startsWith(char *str2)
{
        int lenstr2 = strlen(str2);
        if (!strncmp(this->sval, str2, lenstr2)) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imite endsWith de java.lang.String
bool String::endsWith(String str2)
{
        // str2 doit être plus courte que la chaîne courante
```

```
        if (strlen(str2.sval) > strlen(sval))
                return false;

        if (!strncmp(& this->sval[strlen(sval) - strlen(str2.sval)], str2.sval, strlen(str2.sval) )) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imite endsWith de java.lang.String
bool String::endsWith(char *str2)
{
        // str2 doit être plus courte que la chaîne courante
        if (strlen(str2) > strlen(sval))
                return false;

        if (!strncmp(& this->sval[strlen(sval) - strlen(str2)], str2, strlen(str2) ) ) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imite compareTo de java.lang.String
// Pour les tris, vous devez savoir si l'un est plus petit, égal ou plus grand que l'autre.
// Une chaîne est plus petite qu'une autre si elle arrive avant l'autre dans l'ordre
// lexicographique. Un chaîne est plus grande qu'une autre si elle arrive après.
//  Négatif  --> la chaîne courante est plus petite que str2
//  Positif  --> la chaîne courante est plus grande que str2
//  Zero  --> les deux chaînes sont égales
int String::compareTo(String str2)
{
        int  flag = 0;
        // Compare les lettres dans la chaîne à chaque lettre de str2
        for (int tmpii = 0, tmpjj = strlen(sval), tmpkk = strlen(str2.sval); tmpii < tmpjj; tmpii++)
        {
                if (tmpii > tmpkk)
                        break;
                if (sval[tmpii] == str2.sval[tmpii])
                        flag = 0;
                else
                if (sval[tmpii] > str2.sval[tmpii])
                {
                        flag = 1;
                        break;
                }
                else // if (sval[tmpii] < str2.sval[tmpii])
                {
                        flag = -1;
                        break;
                }
```

```
        }
        return flag;
}

// Imite compareTo de java.lang.String
// Fonction surchargée de compareTo
int String::compareTo(char *str2)
{
        int  flag = 0;
        // Compare les lettres de la chaîne courante avec chaque lettre de str2
        for (int tmpii = 0, tmpjj = strlen(sval), tmpkk = strlen(str2); tmpii < tmpjj; tmpii++)
        {
                if (tmpii > tmpkk)
                        break;
                if (sval[tmpii] == str2[tmpii])
                        flag = 0;
                else
                if (sval[tmpii] > str2[tmpii])
                {
                        flag = 1;
                        break;
                }
                else // if (sval[tmpii] < str2[tmpii])
                {
                        flag = -1;
                        break;
                }
        }
        return flag;
}

// Imite compareToIgnoreCase de java.lang.String
int String::compareToIgnoreCase(String str2)
{
        String tmpaa = this->toLowerCase(),
        tmpbb = str2.toLowerCase();

        return tmpaa.compareTo(tmpbb);
}

// Imite compareToIgnoreCase de java.lang.String
// Version surchargée
int String::compareToIgnoreCase(char *str2)
{
        String tmpaa = this->toLowerCase(),
        tmpcc(str2), tmpbb = tmpcc.toLowerCase();

        return tmpaa.compareTo(tmpbb);
}
```

```
// Imite indexOf de java.lang.String
// Cherche la premiere occurrence d'un caractère ou d'une chaîne.
// Retourne l'indice à partir duquel le caractère ou la chaîne
// a été trouvé, ou -1 en cas d'échec.
int String::indexOf(char ch, int startIndex = 0)
{
        verifyIndex(startIndex);
        int ii = startIndex;
        for (; ii < (int) strlen(sval); ii++)
        {
                if (sval[ii] == ch)
                        break;
        }
        if (ii == (int) strlen(sval))
                return -1;
        return ii;
}

// Imite indexOf de java.lang.String
// Version surchargée
int String::indexOf(char *str2, int startIndex = 0)
{
        verifyIndex(startIndex);
        char * tok;
        long res = -1;

        if ( !isNull() )
        {
                tok = strstr(sval + startIndex, str2);
                if (tok == NULL)
                        res = -1;
                else
                        res = (int) (tok - sval);
        }
        return res;
}

// Imite indexOf de java.lang.String
// Version surchargée
int String::indexOf(String str2, int startIndex = 0)
{
        verifyIndex(startIndex);
        char * tok;
        long res = -1;

        if ( !isNull() )
        {
                tok = strstr(sval + startIndex, str2.sval);
                if (tok == NULL)
                        res = -1;
```

```
                else
                        res = (int) (tok - sval);
        }
        return res;
}

// Imite lastIndexOf de java.lang.String
// Cherche pour la dernière occurrence d'un caractère ou d'une chaîne.
// Retourne l'indice à partir duquel le caractère ou la chaîne a été trouvé
// ou -1 en cas d'échec.
int String::lastIndexOf(char ch, int startIndex = 0)
{
        verifyIndex(startIndex);
        int ii;

        // Commence la recherche par le dernier caractère de la chaîne
        if (!startIndex) // if (startIndex == 0)
                ii = strlen(sval);
        else
                ii = startIndex;
        for (; ii > -1; ii--)
        {
                if (sval[ii] == ch)
                        break;
        }
        if (!ii && sval[ii] != ch) // if (ii == 0)
                return -1;
        return ii;
}

// Imite lastIndexOf de java.lang.String
// Version surchargée
int String::lastIndexOf(char *str2, int startIndex = 0)
{
        verifyIndex(startIndex);
    char *tok = NULL;
    int res = -1;

    register char *tmpaa = strdup(sval);  // ici malloc
    if (!tmpaa) // tmpaa == NULL
    {
        cerr << "\nMemory alloc failed in strdup in lastIndexOf()\n" << endl;
        exit(-1);
    }

    if (!startIndex) // if (startIndex == 0)
        startIndex = strlen(sval);
    else
        tmpaa[startIndex+1] = 0;
```

```
    for (int ii = 0; ii <= startIndex; ii++)
    {
        tok = strstr(& tmpaa[ii], str2);
        if (tok == NULL)
            break;
        else
        {
            res = (int) (tok - tmpaa);
                        debug_("res", res);
            ii = res; // saute vers l'endroit qui correspond (+1 dans la boucle for)
        }
    }
    free(tmpaa);
        debug_("res", res);
        debug_("indexOf", & sval[res]);

        return res;
}

// Imite lastIndexOf de java.lang.String
// Version surchargée
int String::lastIndexOf(String str2, int startIndex = 0)
{
        verifyIndex(startIndex);
    char *tok = NULL;
    int res = -1;

    register char *tmpaa = strdup(sval);  // ici malloc
    if (!tmpaa) // tmpaa == NULL
    {
        cerr << "\nMemory alloc failed in strdup in lastIndexOf()\n" << endl;
        exit(-1);
    }

    if (!startIndex) // if (startIndex == 0)
        startIndex = strlen(sval);
    else
        tmpaa[startIndex+1] = 0;

    for (int ii = 0; ii <= startIndex; ii++)
    {
        tok = strstr(& tmpaa[ii], str2.sval);
        if (tok == NULL)
            break;
        else
        {
            res = (int) (tok - tmpaa);
                        debug_("res", res);
            ii = res; // saute vers l'endroit qui correspond (+1 dans la boucle for)
        }
```

```
    }
    free(tmpaa);
        debug_("res", res);
        debug_("indexOf", & sval[res]);

        return res;
}

// Imite substring de java.lang.String
// startIndex spécifie l'indice de début, et endIndex l'indice de fin.
// La chaîne retournée contient tous les caractères de l'indice de début
// jusqu'à l'indice de fin, mais sans l'inclure.
String String::substring(int startIndex, int endIndex = 0)
{
        String tmpstr = String(sval);
        tmpstr._substring(startIndex, endIndex);
        return tmpstr;
}

// Imite concat de java.lang.String
String String::concat(String str2)
{
        return (*this + str2);
}

// Imite concat de java.lang.String
// Version surchargée
String String::concat(char *str2)
{
        return (*this + str2);
}

// Imite replace de java.lang.String
// Remplace toutes les occurrences de la chaîne 'original' par
// 'replacement' dans 'sval'
String String::replace(char original, char replacement)
{
        // Par exemple -
        //              replace('A', 'B') dans sval = "des AAA et AAACC"
        //              retourne sval = "des BBB et BBBCC"
        //String *tmpstr = new String(sval); Utilise le constructeur de recopie par défaut
        String tmpstr(sval);
        for (int ii = 0, len = strlen(sval); ii < len; ii++)
        {
                if (tmpstr.sval[ii] == original)
                        tmpstr.sval[ii] = replacement;
        }
        return tmpstr; // utilise le constructeur de recopie pour faire une copie
}
```

```
// Imite replace de java.lang.String
// Version surchargée
// Remplace toutes les occurrences de la chaîne 'original' par
// 'replacement' dans 'sval'
String String::replace(char *original, char *replacement)
{
        char *tok = NULL, *bb;
        register char *aa = strdup(sval);
        int lenrepl = strlen(replacement);

        // Alloue l'espace pour bb
        { // portée locale
                int tmpii = 0;
                for (int ii = 0; ;ii++)
                {
                        tok = strstr(& aa[ii], original);
                        if (tok == NULL)
                                break;
                        else
                        {
                                ii = ii + (int) (tok -aa);
                                tmpii++;
                        }
                }
                if (!tmpii) // tmpii == 0, pas de 'original' trouvé
                        return (String(sval)); // retourne la chaîne originale
                tmpii = strlen(sval) + (tmpii * lenrepl) + 20;
                debug_("strstr tmpii", tmpii );
                bb = (char *) malloc(tmpii);
                memset(bb, 0, tmpii);
        }

        for (int res = -1; ;)
        {
                debug_("aa", aa);
                tok = strstr(aa, original);
                if (tok == NULL)
                {
                        strcat(bb, aa);
                        break;
                }
                else
                {
                        res = (int) (tok - aa);
                        strncat(bb, aa, res);
                        strcat(bb, replacement);
                        //bb[strlen(bb)] = 0;
                        debug_("res", res );
                        debug_("bb", bb );
                        strcpy(aa, & aa[res+lenrepl]);
```

```
                }
        }
        debug_("bb", bb );
        free(aa);
        String tmpstr(bb);
        free(bb);
        return tmpstr;
}
/*
une autre méthode pour faire le remplacement mais lente...
String String::replace(char *original, char *replacement)
{
        // Par exemple -
        //              replace("AAA", "BB") avec sval = "des AAA et AAACC"
        //              retourne sval = "des BB et BBCC"
        String bb(this->before(original).sval);
        if (strlen(bb.sval) == 0)
                return String(sval); // retourne la chaîne originale
        bb += replacement;

        String tmpaa(this->sval), cc, dd;
        for (;;)
        {
                cc = tmpaa.after(original).sval;
                debug_("cc", cc.sval );
                if (!strlen(cc.sval)) // if (strlen(cc.sval) == 0)
                        break;

                dd = cc.before(original).sval;
                if (strlen(dd.sval) == 0)
                {
                        bb += cc;
                        break;
                }
                else
                {
                        bb += dd;
                        bb += replacement;
                }
                tmpaa = cc;
        }
        debug_("bb.sval", bb.sval );
        return bb;
}
*/

// Imite replace de Java -  StringBuffer
String String::replace (int startIndex, int endIndex, String str)
{
        verifyIndex(startIndex);
```

```
        verifyIndex(endIndex);
        int tmpjj = strlen(str.sval);
        if (tmpjj == 0)
                return *this;
        int tmpii = endIndex-startIndex-1;
        if (tmpjj < tmpii) // la longueur de str est plus petite que les indices specifies.
                tmpii = tmpjj;
        debug_("sval", sval);
        debug_("str.sval", str.sval);
        strncpy(& sval[startIndex], str.sval, tmpii);
        sval[startIndex+tmpii] = 0;
        debug_("sval", sval);
        return *this;
}

// Imite trim de java.lang.String
String String::trim()
{
        //String *tmpstr = new String(sval);
        String tmpstr(sval);
        tmpstr._trim();
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr; // utilise le constructeur par recopie pour faire une copie
}

// Imite insert de java.lang.String
String String::insert(int index, String str2)
{
        String tmpstr(this->insert(str2.sval, index).sval);
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr;
}

// Imite insert de java.lang.String
String String::insert(int index, char ch)
{
        char aa[2];
        aa[0] = ch;
        aa[1] = 0;
        String tmpstr(this->insert(aa, index).sval);
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr;
}

// Imite deleteCharAt de java.lang.String
String String::deleteCharAt(int loc)
{
        String tmpstr(sval);
        tmpstr._deleteCharAt(loc);
        return tmpstr;
```

```
}

// Imite delete de java.lang.String
// Note : -->le nom Java est "delete()", mais c'est un mot résérvé inutilisable en C++
// startIndex spécifie l'indice du premier caractère à enlever,
// et endIndex l'indice juste après le dernier caractère à supprimer.
// Ainsi, la sous-chaîne supprimée s'etend de startIndex à (endIndex - 1).
String String::deleteStr(int startIndex, int endIndex)
{
        // Par exemple -
        //      deleteStr(3,3) avec val = 'pokemon' retourne 'poon'
        String tmpstr(sval);
        tmpstr._deleteStr(startIndex, endIndex);
        return tmpstr;
}

// Imite reverse de java.lang.String
String String::reverse()
{
        // Par exemple :
        //              reverse() sur "12345" retourne "54321"
        String tmpstr(sval);
        tmpstr._reverse();
        return tmpstr;
}

// Imite valueOf de java.lang.String
String String::valueOf(char chars[], int startIndex, int numChars)
{
        verifyIndex(startIndex);
        int ii = strlen(chars);
        if (startIndex > ii)
        {
                cerr << "\nvalueOf() - startIndex greater than string length of"
                        << "string passed" << endl;
                exit(0);
        }
        if ( (numChars+startIndex) > ii)
        {
                cerr << "\nvalueOf() - numChars exceeds the string length of"
                        << "string passed" << endl;
                exit(0);
        }

        char *aa = strdup(chars);
        aa[startIndex + numChars] = 0;
        String tmpstr(& aa[startIndex]);
        free(aa);
        return tmpstr;
}
```

```
// Imite ensureCapacity de java.lang.String
// Utilisé par la classe StringBuffer.
// Pré-alloue la place pour un certain nombre de caractères.
// Utile si vous savez à l'avance que vous allez rajouter un grand nombre
// de petites chaînes à un StringBuffer
void String::ensureCapacity(int capacity)
{
        sval = (char *) my_realloc(sval, capacity);
        sval[0] = '\0';
        debug_("In ensureCapacity(int capacity) sval", sval);
}

// Imite setLength de java.lang.String
// Utilise par la classe StringBuffer.
void String::setLength(int len)
{
        sval = (char *) my_realloc(sval, len);
        sval[0] = '\0';
        debug_("In ensureCapacity(int len) sval", sval);
}

// Imite setCharAt de StringBuffer
void String::setCharAt(int where, char ch)
{
        verifyIndex(where);
        sval[where] = ch;
        debug_("in StringBuffer dstr()", "ok");
}

// ---- Fin des fonctions imitant java.lang.String -----

// Version surchargée, modifie directement l'objet
// La variable dummy donne une signature différente à la fonction.
void String::substring(int startIndex, int endIndex, bool dummy)
{
        this->_substring(startIndex, endIndex);
}

inline void String::_substring(int startIndex, int endIndex)
{
        verifyIndex(startIndex);
        verifyIndex(endIndex);
        if (!endIndex) // endIndex == 0
                strcpy(sval, & sval[startIndex] ) ;
        else
        {
                if (endIndex > startIndex)
                {
                        strcpy(sval, & sval[startIndex] ) ;
```

```
                        sval[endIndex -startIndex] = 0;
                }
                else
                {
                        cerr << "\n_substring() - startIndex is greater than endIndex!!\n"
                                << endl;
                        exit(-1);
                }
        }
}

// Version surchargée, modifie directement l'objet
String String::deleteStr(int startIndex, int endIndex, bool dummy)
{
        this->_deleteStr(startIndex, endIndex);
        return *this;
}

inline void String::_deleteStr(int startIndex, int endIndex)
{
        verifyIndex(startIndex);
        verifyIndex(endIndex);
        // Par exemple -
        //      deleteStr(3,3) avec val = 'pokemon' retourne 'poon'
        char *tmpaa = strdup(sval); // ici malloc
        strcpy(& tmpaa[startIndex], & tmpaa[endIndex]);
        *this = tmpaa;
        free(tmpaa);
}

// Version surchargée, modifie directement l'objet
String String::deleteCharAt(int loc, bool dummy)
{
        this->_deleteCharAt(loc);
        return *this;
}

inline void String::_deleteCharAt(int loc)
{
        char *tmpaa = strdup(sval); // ici malloc
        strcpy(& tmpaa[loc], & tmpaa[loc+1]);
        *this = tmpaa;
        free(tmpaa);
}

// Retourne la chaîne avant regx. Trouve la première occurrence de regx.
String String::at(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
```

```
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok - sval);
                char *lefttok = strdup(sval);
                memset(lefttok, 0, strlen(sval));
                strcpy(lefttok, & sval[res]);
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}

// Retourne la chaîne avant regx. Trouve la première occurrence de regx.
String String::before(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok - sval);
                char *lefttok = strdup(sval);
                lefttok[res] = 0;
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}

// Retourne la chaîne après regx. Trouve la première occurrence de regx.
String String::after(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok - sval);
                char *lefttok = strdup(sval);
                memset(lefttok, 0, strlen(sval));
                strcpy(lefttok, & sval[res + strlen(regx)]);
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}
```

```
// Divise la chaîne et retourne une liste via le pointeur de tête
// de liste explodeH.
// Voir aussi token().
void String::explode(char *seperator)
{
        char *aa = NULL, *bb = NULL;
        aa = (char *) my_malloc(strlen(sval));
        for (bb = strtok(aa, seperator); bb != NULL; bb = strtok(NULL, seperator) )
        {
                String *tmp = new String(bb);
                String::explodeH.insert(String::explodeH.end(), *tmp);
        }
        my_free(aa);

        list<String>::iterator iter1; // voir include/g++/stl_list.h
        debug_("Before checking explode..", "ok");
        if (String::explodeH.empty() == true )
        {
                debug_("List is empty!!", "ok");
        }

        for (iter1 = String::explodeH.begin(); iter1 != String::explodeH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        debug_("Iterator iter1 is NULL!!", "ok" );
                        break;
                }
                debug_("(*iter1).sval", (*iter1).sval);
        }
}

// Version surchargée de explode(). Retourne un tableau
// de chaînes et le nombre total dans la référence strcount
// Voir aussi token().
String *String::explode(int & strcount, char seperator = ' ')
{
        String aa(sval);
        aa.trim(true);
        strcount = 0;
        for (int ii = 0, jj = strlen(aa.sval); ii < jj; ii++)
        {
                if (aa.sval[ii] == seperator)
                        strcount++;
        }

        String *tmpstr = new String[strcount+1];
        if (!strcount) // strcount == 0
                tmpstr[0] = aa.sval;
```

```
        else
        {
                for (int ii = 0; ii <= strcount; ii++)
                        tmpstr[ii] = aa.token();
        }
        return tmpstr;
}

// Agrège les chaînes pointées par la tête de liste
// explodeH et retourne la classe String.
void String::implode(char *glue)
{
}

// Agrège les chaînes pointées par la tête de liste
// explodeH et retourne la classe String.
void String::join(char *glue)
{
        implode(glue);
}

// Répète la chaîne input n fois.
String String::repeat(char *input, unsigned int multiplier)
{
        // Pour exemple -
        // repeat("k", 4) retourne "kkkk"
        if (!input) // input == NULL
        {
                return (String(""));
        }

        char *aa = (char *) my_malloc(strlen(input) * multiplier);
        for (unsigned int tmpii = 0; tmpii < multiplier; tmpii++)
        {
                strcat(aa, input);
        }
        String tmpstr(aa);
        my_free(aa);
        return tmpstr;
}

// Renverse la chaîne.
// Version surchargée de reverse(). Modifie directement l'objet.
void String::reverse(bool dummy)
{
        this->_reverse();
}
inline void String::_reverse()
{
        // Par exemple -
```

```
        //              reverse() sur "12345" retourne "54321"
        char aa;
        unsigned long tot_len = strlen(sval);
        unsigned long midpoint = tot_len / 2;
        for (unsigned long tmpjj = 0; tmpjj < midpoint; tmpjj++)
        {
                aa = sval[tmpjj];  // variable temporaire de stockage
                sval[tmpjj] = sval[tot_len - tmpjj - 1];  // permute les valeurs
                sval[tot_len - tmpjj - 1] = aa; // permute les valeurs
        }
}

// Change certain caractères.
// Par exemple ("abcd", "ABC") change toutes les occurrences de chaque
// caractère de 'from' en le caractère correspondant dans 'to'
String String::tr(char *from, char *to)
{
        int lenfrom = strlen(from), lento = strlen(to);
        if (lento > lenfrom)
                lento = lenfrom; // choisit le min
        else
        if (lento < lenfrom)
                lenfrom = lento; // choisit le min
        debug_("lento", lento);

        register char *aa = strdup(sval);
        for (int ii = 0, jj = strlen(sval); ii < jj; ii++) // pour chaque caractère dans val
        {
                for (int kk = 0; kk < lento; kk++) // pour chaque caractère dans "from"
                {
                        if (aa[ii] == from[kk])
                                aa[ii] = to[kk];
                }
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Centre le texte
String String::center(int padlength, char padchar = ' ')
{
        // Par exemple -
        //              center(10, '*') avec sval="aa" retourne "****aa****"
        //              center(10) avec sval="aa" retourne "    aa    "
        // Le résultat est une chaîne contenant 'padlength' caractères avec sval au milieu.
        int tmpii = sizeof(char) * (padlength + strlen(sval) + 10);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
```

```
        for (int jj = 0, kk = (int) padlength/2; jj < kk; jj++)
        {
                aa[jj] = padchar;
        }
        strcat(aa, sval);
        for (int jj = strlen(aa), kk = jj + (int) padlength/2; jj < kk; jj++)
        {
                aa[jj] = padchar;
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Formate la chaîne originale en plaçant <number> caractères <padchar>
// entre chaque ensemble de mots délimités par des blancs. Les blancs en début et
// en fin sont toujours supprimés. Si <number> est omis ou vaut 0, alors tous les
// espaces de la chaîne sont supprimés. Par défaut, <number> vaut 0 et padchar ' '.
String String::space(int number, char padchar = ' ')
{
        // Par exemple -
        //              space(3) avec sval = "Je ne sais pas"
        //                              retournera "Je   ne   sais   pas"
        //              space(1, '_') avec sval = "Un lieu profondement obscur"
        //                              retournera "Un_lieu_profondement_obscur"
        //              space() avec sval = "Je   sais     cela"
        //                              retournera "Jesaiscela"

        debug_("this->sval", this->sval );
        String tmpstr = this->trim().sval;
        debug_("tmpstr.sval", tmpstr.sval );

        // compte les espaces
        int spacecount = 0;
        for (int ii = 0, jj = strlen(tmpstr.sval); ii < jj; ii++)
        {
                if (tmpstr.sval[ii] == ' ')
                        spacecount++;
        }
        debug_("spacecount", spacecount);

        char ee[2];
        ee[0] = padchar;
        ee[1] = 0;
        String bb = tmpstr.repeat(ee, spacecount);

        int tmpii = sizeof(char) * (strlen(tmpstr.sval) + (number * spacecount) + 20);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        for (int ii = 0, jj = strlen(tmpstr.sval); ii < jj; ii++)
```

```
        {
                if (tmpstr.sval[ii] == ' ')
                        strcat(aa, bb.sval);
                else
                {
                        ee[0] = sval[ii];
                        strcat(aa, ee);
                }
        }
        tmpstr = aa;
        free(aa);
        return tmpstr;
}

// Le résultat est une chaîne comprenant tous les caractères compris
// entre <start> et <end> (inclus).
String String::xrange(char start, char end)
{
        // Par exemple -
        //      xrange('a', 'j') retourne val = "abcdefghij"
        //      xrange(1, 8) retourne val = "12345678"

        if (end < start)
        {
                cerr << "\nThe 'end' character is less than 'start' !!" << endl;
                return String("");
        }

        // Note : 'end' est plus grand que 'start' ! Et ajoute +1
        int tmpii = sizeof(char) * (end - start + 11);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        debug_("xrange tmpii", tmpii);
        for (int ii = start, jj = 0; ii <= end; ii++, jj++)
        {
                aa[jj] = ii;
                debug_("xrange aa[jj]", aa[jj] );
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Supprime tous les caractères contenus dans <list>. Le caractère par défaut pour
// <list> est l'espace ' '.
String String::compress(char *list = " ")
{
        // Par exemple -
        //      compress("$,%") avec sval = "$1,934" retourne "1934"
        //      compress() avec sval = "appelez moi alavoor vasudevan" returns "appelezmoialavoorvasudevan"
```

```
        int lenlist = strlen(list);
        register char *aa = strdup(sval);
        for (int ii = 0, jj = strlen(sval); ii < jj; ii++) // pour chaque caractère de sval
        {
                for (int kk = 0; kk < lenlist; kk++) // pour chaque caractère de "from"
                {
                        if (aa[ii] == list[kk])
                        {
                                strcpy(& aa[ii], & aa[ii+1]);
                        }
                }
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// <newstr> est insérée dans sval à partir de <start>. <newstr> est
// complétée ou tronquée à <length> caractères. <length> est par défaut la
// longueur de la chaîne <newstr>
String String::insert(char *newstr, int start = 0, int lengthstr = 0, char padchar = ' ')
{
        // Par exemple -
        //      insert("quelquechose de nouveau", 8, 30, '*') avec sval = "vieille chose"
        //              retourne "vieille quelquechose de nouveau*******chose"

        int tmplen = sizeof(char) * strlen(sval) + strlen(newstr) + lengthstr + 10;
        char *tmpaa = (char *) malloc (tmplen);
        memset(tmpaa, 0, tmplen);
        if (!start) // start == 0
        {
                strcpy(tmpaa, newstr);
                strcat(tmpaa, this->sval);
        }
        else
        {
                strncpy(tmpaa, this->sval, start);
                strcat(tmpaa, newstr);
                strcat(tmpaa, & this->sval[start]);
        }

        String tmpstr(tmpaa);
        free(tmpaa);
        return tmpstr;
}

// Fonction insert surchargée
String String::insert(int index, String str2, bool dummy)
{
        *this =  this->insert(str2.sval, index).sval;
```

```
        //debug_("tmpstr.sval", tmpstr.sval);
        return *this;
}

// Fonction insert surchargée
String String::insert(int index, char ch, bool dummy)
{
        char aa[2];
        aa[0] = ch;
        aa[1] = 0;
        *this = this->insert(aa, index).sval;
        //debug_("tmpstr.sval", tmpstr.sval);
        return *this;
}

// Le résultat est une chaîne de <length> caractères composée des caractères les plus à gauches de sval.
// Méthode rapide pour justifier à gauche une chaîne.
String String::left(int slength = 0, char padchar = ' ')
{
        // Par exemple -
        //      left(15) avec sval = "Wig"       retourne "Wig            "
        //      left(4) avec  sval = "Wighat"    retourne "Wigh"
        //      left() avec   sval = "   Wighat" retourne "Wighat   "
        if (!slength) // slength == 0
                slength = strlen(sval);
        debug_("left() slength", slength);

        int tmpii = slength + 20;
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        debug_("this->ltrim().sval ", this->ltrim().sval);
        strcpy(aa, this->ltrim().sval);
        debug_("left() aa", aa );

        int currlen = strlen(aa);
        if (currlen < slength)
        {
                // pad the string now
                char ee[2];
                ee[0] = padchar;
                ee[1] = 0;
                strcat(aa, this->repeat(ee, (unsigned int) (slength-currlen) ).sval);
        }
        else
        {
                aa[slength] = 0;
        }

        debug_("left() aa", aa );
        String tmpstr(aa);
```

```
        free(aa);
        return tmpstr;
}

// Le résultat est une chaîne de <length> caractères composée des caractères les plus à droite de sval.
// Méthode rapide pour justifier à droite un chaîne.
String String::right(int slength = 0, char padchar = ' ')
{
        // Par exemple -
        //      right(10) avec sval = "never to saying   " retourne " to saying"
        //      right(4) avec  sval = "Wighat"             retourne "ghat"
        //      right(8) avec  sval = "4.50"               retourne "    4.50"
        //      right() avec   sval = "  4.50     "        retourne "       4.50"

        if (!slength) // slength == 0
                slength = strlen(sval);
        debug_("right() slength", slength);

        int tmpii = slength + 20;
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);

        int currlen = strlen(this->rtrim().sval);
        debug_("right() currlen", currlen );
        if (currlen < slength)
        {
                // pad the string now
                char ee[2];
                ee[0] = padchar;
                ee[1] = 0;
                strcpy(aa, this->repeat(ee, (unsigned int) (slength-currlen) ).sval);
                strcat(aa, this->rtrim().sval);
                debug_("right() aa", aa );
        }
        else
        {
                strcpy(aa, this->rtrim().sval);
                strcpy(aa, & aa[currlen-slength]);
                aa[slength] = 0;
        }

        debug_("right() aa", aa );
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// <newstr> est superposée à sval en commençant à <start>. <newstr> est complétée
// ou tronquée à <length> caractères. Par défaut, la longueur <length> est la
// longueur de la chaîne newstr.
```

```
String String::overlay(char *newstr, int start = 0, int slength = 0, char padchar = ' ')
{
	// Par exemple -
	//	overlay("12345678", 4, 10, '*') sur sval = "oldthing is very bad"
	//		retourne "old12345678**ery bad"
	//	overlay("12345678", 4, 5, '*') sur sval = "oldthing is very bad"
	//		retourne "old12345ery bad"
	int len_newstr = strlen(newstr);
	if (!slength) // slength == 0
		slength = len_newstr;
	char *aa = (char *) malloc(slength + len_newstr + 10);
	aa[0] = 0;
	char ee[2];
	ee[0] = padchar;
	ee[1] = 0;
	if (len_newstr < slength)
	{
		// remplire maintenant
		strcpy(aa, newstr);
		strcat(aa, this->repeat(ee, (slength-len_newstr)).sval );
	}
	else
	{
		strcpy(aa, newstr);
		aa[slength] = 0;
	}

	// Maintenant recouvrir la chaîne
	String tmpstr(sval);

	debug_("tmpstr.sval", tmpstr.sval);
	for (int ii=start, jj=strlen(tmpstr.sval), kk=start+slength, mm=0;
				ii < jj; ii++, mm++)
	{
		if (ii == kk)
			break;
		if (mm == slength)
			break;
		tmpstr.sval[ii] = aa[mm];
	}
	free(aa);
	debug_("tmpstr.sval", tmpstr.sval);
	return tmpstr;
}

// Si la chaîne est littéralement égale à ou non égale à
// Si type vaut false alors ==
bool String::_equalto(const String & rhs, bool type = false)
{
	if (type == false) // test ==
```

```
        {
                if (strlen(rhs.sval) == strlen(sval))
                {
                        if (!strncmp(rhs.sval, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
        else // test !=
        {
                if (strlen(rhs.sval) != strlen(sval))
                {
                        if (!strncmp(rhs.sval, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
}

// Si la chaîne est littéralement égale à ou non égale à
// Si type vaut false alors ==
bool String::_equalto(const char *rhs, bool type = false)
{
        if (type == false) // test ==
        {
                if (strlen(rhs) == strlen(sval))
                {
                        if (!strncmp(rhs, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
        else // test !=
        {
                if (strlen(rhs) != strlen(sval))
                {
                        if (!strncmp(rhs, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
```

```
                else
                        return false;
        }
}

// Fonction synonyme de vacuum()
void String::clear()
{
        sval = (char *) my_realloc(sval, 10);
        sval[0] = '\0';
}

// Supprime tous les caractères 'ch' en fin de chaîne - voir aussi chop()
// Par exemple :
//      sval = "abcdef\n\n\n" alors chopall() = "abcdef"
//      sval = "abcdefffff" alors chopall('f') = "abcde"
void String::chopall(char ch='\n')
{
        unsigned long tmpii = strlen(sval) - 1 ;
        for (; tmpii >= 0; tmpii--)
        {
                if (sval[tmpii] == ch)
                        sval[tmpii] = 0;
                else
                        break;
        }
}

// Supprime le caractère de fin de la chaîne - voir aussi chopall()
// chop() est souvent utilisé pour supprimer le caractère de fin de ligne
void String::chop()
{
        sval[strlen(sval)-1] = 0;
}

// Version surchargée de trim(). Modifie directement l'objet.
void String::trim(bool dummy)
{
        this->_trim();
}

inline void String::_trim()
{
        this->rtrim(true);
        this->ltrim(true);
        debug_("this->sval", this->sval);
}

// Version surchargée de ltrim(). Modifie directement l'objet.
void String::ltrim(bool dummy)
```

```
{
        this->_ltrim();
}

inline void String::_ltrim()
{
        // Peut causer des problèmes dans my_realloc car
        // l'emplacement de bb peut être détruit !
        char *bb = sval;

        if (bb == NULL)
                return;

        while (isspace(*bb))
                bb++;
        debug_("bb", bb);

        if (bb != NULL && bb != sval)
        {
                debug_("doing string copy", "done");
                _str_cpy(bb); // cause des problèmes dans my_realloc et bb va être détruit !
        }
        else
                debug_("Not doing string copy", "done");
}

String String::ltrim()
{
        String tmpstr(sval);
        tmpstr._ltrim();
        return tmpstr;
}

// Version surchargée de rtrim(). Modifie directement l'objet.
void String::rtrim(bool dummy)
{
        this->_rtrim();
}

inline void String::_rtrim()
{
        for (long tmpii = strlen(sval) - 1 ; tmpii >= 0; tmpii--)
        {
                if ( isspace(sval[tmpii]) )
                        sval[tmpii] = '\0';
                else
                        break;
        }
}
```

```
String String::rtrim()
{
        String tmpstr(sval);
        tmpstr._rtrim();
        return tmpstr;
}

// Utilisé pour arrondir la partie frationnaire de réels.
// Arrondit les réels avec la précision souhaitée et stocke
// le résultat dans le champ sval de la chaîne.
// Retourne aussi le résultat comme un char *.
void String::roundf(float input_val, short precision)
{
        float   integ_flt, deci_flt;
        const   short MAX_PREC = 4;

        debug_("In roundf", "ok");

        if (precision > MAX_PREC) // précision maximale supportée
                precision = MAX_PREC;

        // récupère les parties entière et décimale du réel
        deci_flt = modff(input_val, & integ_flt);

        for (int tmpzz = 0; tmpzz < precision; tmpzz++)
        {
                debug_("deci_flt", deci_flt);
                deci_flt *= 10;
        }
        debug_("deci_flt", deci_flt);

        unsigned long deci_int = (unsigned long) ( rint(deci_flt) );

        sval = (char *) my_malloc(NUMBER_LENGTH); // float 70 chiffres max

        if (deci_int > 999) // (MAX_PREC) chiffres
                sprintf(sval, "%lu.%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 99) // (MAX_PREC - 1) chiffres
                sprintf(sval, "%lu.0%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9) // (MAX_PREC - 2) chiffres
                sprintf(sval, "%lu.00%lu", (unsigned long) integ_flt, deci_int);
        else
                sprintf(sval, "%lu.00000%lu", (unsigned long) integ_flt, deci_int);
}

void String::roundd(double input_val, short precision)
{
        double  integ_flt, deci_flt;
```

```
        const   short MAX_PREC = 6;

        if (precision > MAX_PREC) // précision maximale supportée
                precision = MAX_PREC;

        debug_("In roundd", "ok");
        // récupère les parties entière et décimale du réel
        deci_flt = modf(input_val, & integ_flt);

        for (int tmpzz = 0; tmpzz < precision; tmpzz++)
        {
                debug_("deci_flt", deci_flt);
                deci_flt *= 10;
        }
        debug_("deci_flt", deci_flt);

        sval = (char *) my_malloc(NUMBER_LENGTH); // double 70 chiffres max

        unsigned long deci_int = (unsigned long) ( rint(deci_flt) );

        if (deci_int > 99999) // (MAX_PREC) chiffres
                sprintf(sval, "%lu.%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9999) // (MAX_PREC - 1) chiffres
                sprintf(sval, "%lu.0%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 999) // (MAX_PREC - 2) chiffres
                sprintf(sval, "%lu.00%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 99) // (MAX_PREC - 3) chiffres
                sprintf(sval, "%lu.000%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9) // (MAX_PREC - 4) chiffres
                sprintf(sval, "%lu.0000%lu", (unsigned long) integ_flt, deci_int);
        else // (MAX_PREC - 5) chiffres
                sprintf(sval, "%lu.00000%lu", (unsigned long) integ_flt, deci_int);
}

// Fournie seulement pour documenter
// Vous devriez utiliser la fonction indexOf()
bool String::contains(char *str2, int startIndex = 0)
{
        // Par exemple -
        //              if (indexOf("ohboy") > -1 )
        //                      cout << "\nString contient 'ohboy'" << endl;
        //              if (indexOf("ohboy") < 0 )
        //                      cout << "\nString NE contient PAS 'ohboy'" << endl;
        //              if (indexOf("ohboy", 4) > -1 )
        //                      cout << "\nString contient 'ohboy'" << endl;
        //              if (indexOf("ohboy", 4) < 0 )
```

```
        //                      cout << "\nString NE contient PAS 'ohboy'" << endl;
        cerr << "\nYou must use indexOf() function instead of contains()\n" << endl;
        exit(-1);
}

// Fonction synonyme de empty()
bool String::isNull()
{
        if (sval[0] == '\0')
                return true;
        else
        {
                if (sval == NULL)
                        return true;
                else
                        return false;
        }
}

// Les caractères blancs de début et de fin sont ignorés.
bool String::isInteger()
{
        String tmpstr(sval);
        tmpstr.trim(true);
        debug_("tmpstr.sval", tmpstr.sval );
        if ( strspn ( tmpstr.sval, "0123456789" ) != strlen(tmpstr.sval) )
                return ( false ) ;
        else
                return ( true ) ;
}

// Fonction surchargée
bool String::isInteger(int pos)
{
        verifyIndex(pos);
        return (isdigit(sval[pos]));
}

// Les caractères blancs de début et de fin sont ignorés.
bool String::isNumeric()
{
        String tmpstr(sval);
        tmpstr.trim(true);
        debug_("tmpstr.sval", tmpstr.sval );
        if ( strspn ( tmpstr.sval, "0123456789.+-e" ) != strlen(tmpstr.sval) )
                return ( false ) ;
        else
                return ( true ) ;
}
```

```
// Fonction surchargée
bool String::isNumeric(int pos)
{
        verifyIndex(pos);
        return (isdigit(sval[pos]));
}

bool String::isEmpty()
{
        if (strlen(sval) == 0)
                return true;
        else
                return false;
}

// Voir aussi explode()
//      Attention : l'objet String est modifié et ne contient plus le token renvoyé.
//                  Il est conseillé de sauvegarder la chaîne originale avant d'appeler
//                  cette fonction, comme par exemple ainsi :
//              String savestr = origstr;
//              String aa, bb, cc;
//              aa = origstr.token();
//              bb = origstr.token();
//              cc = origstr.token();
//
//  Cette méthode retourne le premier token non séparateur (par défaut espace) de la chaîne.
String String::token(char seperator = ' ')
{
        char ee[2];
        ee[0] = seperator;
        ee[1] = 0;
        char *res = strtok(sval, ee);
        if (!res) // if res == NULL
        {
                debug_("token", res);
                debug_("sval", sval);
                return(String(sval));
        }
        else
        {
                String tmpstr(res);

                // Utilise la longueur de la chaîne sval et pas celle de res
                // car strtok() a mis un NULL ('\0') sur l'emplacement et
                // aussi car strtok() ignore les blancs initiaux de sval
                strcpy(sval, & sval[strlen(sval)+1]);
                debug_("token", res);
                debug_("sval", sval);
                return tmpstr;
        }
```

```
}

String String::crypt(char *original, char *salt)
{
	return String("");
}

int String::toInteger()
{
	if ( strlen(sval) == 0 ) {
		cerr << "Cannot convert a zero length string "
		<< " to a numeric" << endl ;
		abort() ;
	}

	if ( ! isInteger() ) {
		cerr << "Cannot convert string [" << sval
		<< "] to an integer numeric string" << endl ;
		abort() ;
	}

	return ( atoi ( sval ) ) ;
}

long String::parseLong()
{
	if ( strlen(sval) == 0 ) {
		cerr << "Cannot convert a zero length string "
		<< " to a numeric" << endl ;
		abort() ;
	}

	if ( ! isInteger() ) {
		cerr << "Cannot convert string [" << sval
		<< "] to an integer numeric string" << endl ;
		abort() ;
	}

	return ( atol ( sval ) ) ;
}

double String::toDouble()
{
	if ( strlen(sval) == 0 ) {
		cerr << "Cannot convert a zero length string "
		<< " to a numeric" << endl ;
		abort() ;
	}

	if ( ! isNumeric() ) {
```

```
                cerr << "Cannot convert string [" << sval
                << "] to a double numeric string" << endl ;
                abort() ;
        }

        double d = atof ( sval ) ;

        return ( d ) ;
}

String String::getline(FILE *infp = stdin)
{
        register char ch, *aa = NULL;

        register const short SZ = 100;
        // Valeur initiale de ii > SZ donc aa est de la mémoire allouée
        register int jj = 0;
        for (int ii = SZ+1; (ch = getc(infp)) != EOF; ii++, jj++)
        {
                if (ii > SZ)  // alloue la memoire en paquets de SZ pour la performance
                {
                        aa = (char *) realloc(aa, jj + ii + 15);  // +15 par sécurité
                        ii = 0;
                }
                if (ch == '\n')  // lit jusqu'à rencontrer une nouvelle ligne
                        break;
                aa[jj] = ch;
        }
        aa[jj] = 0;
        _str_cpy(aa);  // met la valeur dans la chaîne
        free(aa);
        return *this;
}

/*
void String::Format(const char *fmt, ... )
{
        va_list iterator;
        va_start(iterator, fmt );
        va_end(iterator);
}
*/

// contrôle qu'un indice se trouve dans les limites
inline void String::verifyIndex(unsigned long index) const
{
        if (index < 0 || index >= strlen(sval) )
        {
                // throw "Index Out Of Bounds Exception";
                cerr << "Index Out Of Bounds Exception at ["
```

```
                                << index << "] in:\n" << sval << endl;
                exit(1);
        }
}

// contrôle qu'un indice se trouve dans les limites
inline void String::verifyIndex(unsigned long index, char *aa) const
{
        if (!aa) // aa == NULL
        {
                cerr << "\nverifyIndex(long, char *) str null value\n" << endl;
                exit(-1);
        }
        if (index < 0 || index >= strlen(aa) )
        {
                cerr << "Index Out Of Bounds Exception at ["
                                << index << "] in:\n" << aa << endl;
                exit(1);
        }
}

//////////////////////////////////////////////////////////
// Les fonctions privées commencent à partir d'ici...
//////////////////////////////////////////////////////////
void String::_str_cpy(char bb[])
{
        debug_("In _str_cpy bb", bb);
        if (bb == NULL)
        {
                sval[0] = '\0';
                return;
        }

        unsigned long tmpii = strlen(bb);

        if (tmpii == 0)
        {
                sval[0] = '\0';
                return;
        }

        debug_("In _str_cpy tmpii", tmpii);
        debug_("In _str_cpy sval", sval);
        sval = (char *) my_realloc(sval, tmpii);
        //sval = new char [tmpii + SAFE_MEM_2];
        debug_("In _str_cpy bb", bb);

        strncpy(sval, bb, tmpii);
        debug_("In _str_cpy sval", sval);
        sval[tmpii] = '\0';
```

```
        debug_("In _str_cpy sval", sval);
}

void String::_str_cpy(int bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%d", bb);
        _str_cpy(tmpaa);
}

void String::_str_cpy(unsigned long bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%ld", bb);
        _str_cpy(tmpaa);
}

void String::_str_cpy(float bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%f", bb);
        _str_cpy(tmpaa);
}

void String::_str_cat(char bb[])
{
        unsigned long tmpjj = strlen(bb), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        debug_("sval in _str_cat() ", sval);
        strncat(sval, bb, tmpjj);
}

void String::_str_cat(int bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%d", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
}

void String::_str_cat(unsigned long bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%ld", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
```

```
}

void String::_str_cat(float bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%f", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
}

//////////////////////////////////////////////////////////
// Les opérateurs sont définis à partir d'ici...
//////////////////////////////////////////////////////////
String operator+ (const String & lhs, const String & rhs)
{
        /*****************************************************/
        // Note : pour concaténer deux chaînes, transtyper d'abord
        // en String comme ici :
        //aa = (String) "alkja " + " 99djd " ;
        /*****************************************************/

        String tmp(lhs);
        tmp._str_cat(rhs.sval);
        return(tmp);

        /*
        if (String::_global_String == NULL)
        {
                String::_global_String = new String;
                String::_global_String->_str_cpy(lhs.sval);
                String::_global_String->_str_cat(rhs.sval);
                //return *String::_global_String;
                return String(String::_global_String->val);
        }
        */
        /*
        else
        if (String::_global_String1 == NULL)
        {
                debug_("1)global", "ok" );
                String::_global_String1 = new String;
                String::_global_String1->_str_cpy(lhs.sval);
                String::_global_String1->_str_cat(rhs.sval);
                return *String::_global_String1;
        }
        */
        /*
        else
```

```
        {
                fprintf(stderr, "\nError: cannot alloc _global_String\n");
                exit(-1);
        }
        */

        /*
        String *aa = new String;
        aa->_str_cpy(lhs.sval);
        aa->_str_cat(rhs.sval);
        return *aa;
        */
}

String String::operator+ (const String & rhs)
{
        String tmp(*this);
        tmp._str_cat(rhs.sval);
        debug_("rhs.sval in operator+", rhs.sval );
        debug_("tmp.sval in operator+", tmp.sval );
        return (tmp);
}

// L'utilisation d'une référence accélèrera l'opérateur =
String& String:: operator= ( const String& rhs )
{
        if (& rhs == this)
        {
                debug_("Fatal Error: In operator(=). rhs is == to 'this pointer'!!", "ok" );
                return *this;
        }

        this->_str_cpy(rhs.sval);
        debug_("rhs value", rhs.sval );

        // Libere la memoire des variables globales
        //_free_glob(& String::_global_String);
        //if (String::_global_String == NULL)
                //fprintf(stderr, "\n_global_String is freed!\n");

        //return (String(*this));
        return *this;
}

// L'utilisation d'une référence accélèrera l'opérateur =
String& String::operator+= (const String & rhs)
{
        /*******************************************************/
        // Note : pour concaténer deux chaînes, transtyper d'abord
        // en String comme ici :
```

```
        //aa = (String) "cccc " + " dddd " ;
        /****************************************************/

        if (& rhs == this)
        {
                debug_("Fatal error: In operator+= rhs is equals 'this' ptr", "ok");
                return *this;
        }
        this->_str_cat(rhs.sval);
        return *this;
        //return (String(*this));
}

bool String::operator== (const String & rhs)
{
        return(_equalto(rhs.sval));
}

bool String::operator== (const char *rhs)
{
        return(_equalto(rhs));
}

bool String::operator!= (const String & rhs)
{
        return(_equalto(rhs.sval, true));
}

bool String::operator!= (const char *rhs)
{
        return(_equalto(rhs, true));
}

char String::operator[] (unsigned long Index) const
{
        verifyIndex(Index);
        return sval[Index];
}

char & String::operator[] (unsigned long Index)
{
        verifyIndex(Index);
        return sval[Index];
}

istream & operator >> (istream & In, String & str2)
{
        // alloue la taille max de 2048 caractères
        static char aa[MAX_ISTREAM_SIZE];
```

```
        In >> aa;
        str2 = aa; // affecte aa à la référence
        return In; // retourne un istream
}

ostream & operator << (ostream & Out, const String & str2)
{
        Out << str2.sval;
        return Out;
}

////////////////////////////////////////////////////
//  Imite StringBuffer Object
//      MéThodes de StringBuffer
////////////////////////////////////////////////////

// Imite StringBuffer ; le constructeur par défaut
// (celui sans paramètre) réserve de la place pour 16
// caractères.
StringBuffer::StringBuffer()
        :String()   // appelle le constructeur de la classe de base sans paramètres
{
        debug_("in StringBuffer cstr()", "ok");
}

// Imite StringBuffer
StringBuffer::StringBuffer(int size)
        :String(size, true) // appelle le constructeur de la classe de base sans paramètres
{
        // String(size, true) -- ne pas l'appeler ici dans le corps de la
        // fonction mais durant la phase d'initialisation pour éviter un
        // appel supplémentaire du constructeur par défaut de la classe de
        // base et être plus rapide et plus efficace
        debug_("in StringBuffer cstr(int size)", "ok");
}

// Imite StringBuffer
// appelle le constructeur de la classe de base avec une chaîne en paramètre
StringBuffer::StringBuffer(String str)
        :String(str.val())  // appelle le constructeur de la classe de base
{
        // String(str.val()) -- ne pas l'appeler ici dans le corps de la
        // fonction mais durant la phase d'initialisation pour éviter un
        // appel supplémentaire du constructeur par défaut de la classe de
        // base et être plus rapide et plus efficace
        debug_("in StringBuffer cstr(String str)", "ok");
}

// Imite StringBuffer
StringBuffer::~StringBuffer()
```

```
{
        debug_("in StringBuffer dstr()", "ok");
}

// Imite la classe Float
// appelle le constructeur de la classe de base avec une chaîne en paramètre
Float::Float(String str)
        :String(str.val())  // appelle le constructeur de la classe de base
{
        // String(str.val()) -- ne pas l'appeler ici dans le corps de la
        // fonction mais durant la phase d'initialisation pour éviter un
        // appel supplémentaire du constructeur par défaut de la classe de
        // base et être plus rapide et plus efficace
        debug_("in Float cstr(String str)", "ok");
}

// Imite la classe Double
// appelle le constructeur de la classe de base avec une chaîne en paramètre
Double::Double(String str)
        :String(str.val())  // appelle le constructeur de la classe de base
{
        // String(str.val()) -- ne pas l'appeler ici dans le corps de la
        // fonction mais durant la phase d'initialisation pour éviter un
        // appel supplémentaire du constructeur par défaut de la classe de
        // base et être plus rapide et plus efficace
        debug_("in Double cstr(String str)", "ok");
}

// Imite la classe StringReader
// appelle le constructeur de la classe de base avec une chaîne en paramètre
StringReader::StringReader(String str)
        :String(str.val())  // appelle le constructeur de la classe de base
{
        // String(str.val()) -- ne pas l'appeler ici dans le corps de la
        // fonction mais durant la phase d'initialisation pour éviter un
        // appel supplémentaire du constructeur par défaut de la classe de
        // base et être plus rapide et plus efficace
        debug_("in StringReader cstr(String str)", "ok");
        _curpos = 0;
        _mark_pos = 0;
}

// Imite la méthode read de la classe StringReader
int StringReader::read()
{
        _curpos++;
        if (_curpos > strlen(sval) )
                return -1;
        return sval[_curpos-1];
}
```

```
// Lit des caractères dans une portion d'un tableau
// cbuf est le tampon destination, offset est le décalage indiquant où écrire
// les caractères, length est le nombre maximum de caractères à lire
// Retourne le nombre de caractères lus ou -1 en cas de fin du flux
int StringReader::read(char cbuf[], int offset, int length)
{
        if (_curpos > strlen(sval) - 1 )
                return -1;
        strncpy(& cbuf[offset], & sval[_curpos], length);
        _curpos += length;
        return length;
}

// Marque la position courante dans le flux. Les appels suivants
// à reset() repositionneront le flux à ce point.
// Le paramètre 'readAheadLimit' limite le nombre de caractères qui
// pourraient être lus tout en préservant la marque.
// Comme le flux d'entrée provient d'une chaîne, il n'y a pas de
//limite actuellement, donc l'argument est ignoré.
void StringReader::mark(int readAheadLimit)
{
        _mark_pos = _curpos;
}
// réinitialise le flux à la marque la plus récente, ou au début de
// la chaîne si aucun marque n'a été posée
void StringReader::reset()
{
        _curpos = _mark_pos;
}

// Passe des caractères. Cette méthode bloquera jusqu'a ce que des caractères soient
// disponibles, qu'une erreur arrive ou que la fin du flux soit atteinte.
// Paramètre ii : nombre de caractères à passer
// Retourne : le nombre courant de caractères passés
long StringReader::skip(long ii)
{
        long tmpjj = strlen(sval) - _curpos - 1;
        if (ii > tmpjj)
                ii = tmpjj;
        _curpos = ii;
        return ii;
}

// Imite la classe StringWriter
StringWriter::StringWriter()
{
        debug_("in StringWriter cstr()", "ok");
        char *aa = (char *) malloc(300);
        memset(aa, ' ', 299); // remplit avec des blancs
```

```
        aa[300] = 0;
        String((char *) aa);
        my_free(aa);
}
StringWriter::StringWriter(int bufferSize)
{
        debug_("in StringWriter cstr(int bufferSize)", "ok");
        char *aa = (char *) malloc(bufferSize);
        memset(aa, ' ', bufferSize-1); // remplit avec des blancs
        aa[bufferSize] = 0;
        String((char *) aa);
        my_free(aa);
}
void StringWriter::write(int bb)
{
        _str_cat(bb);
}
void StringWriter::write(char *bb)
{
        _str_cat(bb);
}
void StringWriter::write(String bb)
{
        _str_cat(bb.val());
}
void StringWriter::write(char *bb, int startIndex, int endIndex)
{
        char *aa = strdup(bb); // teste le null dans verifyIndex
        verifyIndex(startIndex, aa);
        verifyIndex(endIndex, aa);
        aa[endIndex] = 0;
        _str_cat(& aa[startIndex]);
}
void StringWriter::write(String str, int startIndex, int endIndex)
{
        write(str.val(), startIndex, endIndex);
}
```

17 Annexe D my_malloc.cpp

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//*****************************************************************
// La licence de distribution est la GNU/GPL et vous devez inclure
// le nom et le mel de l'auteur dans toutes les copies
// Auteur : Al Dev    Mel : alavoor@yahoo.com
//*****************************************************************
```

```
#include <stdio.h>
#include <alloc.h>  // pour malloc, alloc etc...
#include <stdlib.h>  // malloc, alloc..
#include <time.h>  // strftime, localtime, ...
#include <list.h>  // strftime, localtime, ...  voir include/g++/stl_list.h
//#include <debug.h> // debug_("a", a);  debug2_("a", a, true);

#include "my_malloc.h"

const short SAFE_MEM = 10;
const short DATE_MAX_SIZE = 200;

const short MALLOC = 1;
const short REALLOC     = 2;

const short VOID_TYPE =        1;
const short CHAR_TYPE =        2;
const short SHORT_TYPE =       3;
const short INT_TYPE =         4;
const short LONG_TYPE =        5;
const short FLOAT_TYPE =       6;
const short DOUBLE_TYPE =      7;

const char LOG_FILE[30] = "memory_error.log";

// Décommenter cette ligne pour déboguer la totalité de la mémoire allouée
//#define DEBUG_MEM  "debug_memory_sizes_allocated"

static void raise_error_exit(short mtype, short datatype, char fname[], int lineno);

void local_my_free(void *aa, char fname[], int lineno)
{
        if (aa == NULL)
                return;
        //call_free_check(aa, fname, lineno);
        free(aa);
        aa = NULL;
}

// size_t est défini comme un entier long non signé (unsigned long)
void *local_my_malloc(size_t size, char fname[], int lineno)
{
        size_t  tmpii = size + SAFE_MEM;
        void *aa = NULL;
        aa = (void *) malloc(tmpii);
        if (aa == NULL)
                raise_error_exit(MALLOC, VOID_TYPE, fname, lineno);
        memset(aa, 0, tmpii);
        //call_check(aa, tmpii, fname, lineno);
        return aa;
```

```
}

// size_t est défini comme un entier long non signé (unsigned long)
char *local_my_realloc(char *aa, size_t size, char fname[], int lineno)
{
	//remove_ptr(aa, fname, lineno);
	unsigned long tmpjj = 0;
	if (aa) // aa !=  NULL
		tmpjj = strlen(aa);
	unsigned long tmpqq = size + SAFE_MEM;
	size_t  tmpii = sizeof (char) * (tmpqq);
	aa = (char *) realloc(aa, tmpii);
	if (aa == NULL)
		raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);

	// ne pas utiliser memset ! memset(aa, 0, tmpii);
	aa[tmpqq-1] = 0;
	unsigned long kk = tmpjj;
	if (tmpjj > tmpqq)
		kk = tmpqq;
	for ( ; kk < tmpqq; kk++)
		aa[kk] = 0;
	//call_check(aa, tmpii, fname, lineno);
	return aa;
}

// size_t est défini comme un entier long non signé (unsigned long)
short *local_my_realloc(short *aa, size_t size, char fname[], int lineno)
{
	//remove_ptr(aa, fname, lineno);
	unsigned long tmpqq = size + SAFE_MEM;
	size_t  tmpii = sizeof (short) * (tmpqq);
	aa = (short *) realloc(aa, tmpii);
	if (aa == NULL)
		raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
	// ne pas utiliser memset ! memset(aa, 0, tmpii);
	// pas pour les nombres ! aa[tmpqq-1] = 0;
	//call_check(aa, tmpii, fname, lineno);
	return aa;
}

// size_t est défini comme un entier long non signé (unsigned long)
int *local_my_realloc(int *aa, size_t size, char fname[], int lineno)
{
	//remove_ptr(aa, fname, lineno);
	unsigned long tmpqq = size + SAFE_MEM;
	size_t  tmpii = sizeof (int) * (tmpqq);
	aa = (int *) realloc(aa, tmpii);
	if (aa == NULL)
		raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
```

```
        // ne pas utiliser memset ! memset(aa, 0, tmpii);
        // pas pour les nombres ! aa[tmpqq-1] = 0;
        //call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t est défini comme un entier long non signé (unsigned long)
long *local_my_realloc(long *aa, size_t size, char fname[], int lineno)
{
        //remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (long) * (tmpqq);
        aa = (long *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // ne pas utiliser memset ! memset(aa, 0, tmpii);
        // pas pour les nombres ! aa[tmpqq-1] = 0;
        //call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t est défini comme un entier long non signé (unsigned long)
float *local_my_realloc(float *aa, size_t size, char fname[], int lineno)
{
        //remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (float) * (tmpqq);
        aa = (float *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // ne pas utiliser memset ! memset(aa, 0, tmpii);
        // pas pour les nombres ! aa[tmpqq-1] = 0;
        //call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t est défini comme un entier long non signé (unsigned long)
double *local_my_realloc(double *aa, size_t size, char fname[], int lineno)
{
        //remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (double) * (tmpqq);
        aa = (double *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // ne pas utiliser memset ! memset(aa, 0, tmpii);
        // pas pour les nombres ! aa[tmpqq-1] = 0;
        //call_check(aa, tmpii, fname, lineno);
        return aa;
}
```

```
static void raise_error_exit(short mtype, short datatype, char fname[], int lineno)
{
        if (mtype == MALLOC)
        {
                fprintf(stdout, "\nFatal Error: malloc() failed!!");
                fprintf(stderr, "\nFatal Error: malloc() failed!!");
        }
        else
        if (mtype == REALLOC)
        {
                fprintf(stdout, "\nFatal Error: realloc() failed!!");
                fprintf(stderr, "\nFatal Error: realloc() failed!!");
        }
        else
        {
                fprintf(stdout, "\nFatal Error: mtype not supplied!!");
                fprintf(stderr, "\nFatal Error: mtype not supplied!!");
                exit(-1);
        }

        // Récupère la date et l'heure courantes et les met dans le fichier d'erreurs...
        char date_str[DATE_MAX_SIZE + SAFE_MEM];
        time_t tt;
        tt = time(NULL);
        struct tm *ct = NULL;
        ct = localtime(& tt); // time() in secs since Epoch 1 Jan 1970
        if (ct == NULL)
        {
                fprintf(stdout, "\nWarning: Could not find the local time, localtime() failed\n");
                fprintf(stderr, "\nWarning: Could not find the local time, localtime() failed\n");
        }
        else
                strftime(date_str, DATE_MAX_SIZE , "%C", ct);

        FILE *ferr = NULL;
        char    filename[100];
        strcpy(filename, LOG_FILE);
        ferr = fopen(filename, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", filename);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", filename);
        }
        else
        {
                // *************************************************
                // **** Fait le putenv dans la fonction main()  *****
                //              char p_name[1024];
                //              sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
```

```
//              putenv(p_name);
// ************************************************
char    program_name[200+SAFE_MEM];
if (getenv("PROGRAM_NAME") == NULL)
{
        fprintf(ferr, "\n%sWarning: You did not putenv() PROGRAM_NAME env variable in main() function\n",
                        date_str);
        program_name[0] = 0;
}
else
        strncpy(program_name, getenv("PROGRAM_NAME"), 200);

if (mtype == MALLOC)
        fprintf(ferr, "\n%s: %s - Fatal Error - my_malloc() failed.", date_str, program_name);
else
if (mtype == REALLOC)
{
        fprintf(ferr, "\n%s: %s - Fatal Error - my_realloc() failed.", date_str, program_name);
        char dtype[50];
        switch(datatype)
        {
                case VOID_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case CHAR_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case SHORT_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case INT_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case LONG_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case FLOAT_TYPE:
                        strcpy(dtype, "char*");
                        break;
                case DOUBLE_TYPE:
                        strcpy(dtype, "char*");
                        break;
                default:
                        strcpy(dtype, "none*");
                        break;
        }
        fprintf(ferr, "\n%s %s - Fatal Error: %s realloc() failed!!", date_str, program_name, dtype);
}

fprintf(ferr, "\n%s %s - Very severe error condition. Exiting application now....",
```

```
                                        date_str, program_name);
                fclose(ferr);
        }

        exit(-1);
}

// /////////////////////////////////////////////////////////////
//              Fonctions pour afficher la mémoire totale utilisée
//              Ces fonctions peuvent être utilisées pour déboguer
//              Devraient être commentées pour le code de production
// Utilisation de ces fonctions :
//              Dans votre fonction main(), mettre les lignes
//                      char p_name[1024];
//                      sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
//                      putenv(p_name);
//                      print_total_memsize(); // au debut
//                      ......
//                      ......
//                      print_total_memsize(); // à la fin
// /////////////////////////////////////////////////////////////

// *********************************************************
// ** Les fonctions ci-dessus utilisent le conteneur list de la STL.
// ** Pour plus d'aide sur les listes, voir
// **   http://www.sgi.com/Technology/STL and List.html
// **   http://www.sgi.com/Technology/STL and other_resources.html
// **   http://www.halpernwightsoftware.com/stdlib-scratch/quickref.html
// *********************************************************
/*
#ifndef DEBUG
void local_print_total_memsize(char *fname, int lineno)
{
        // Cette fonction est disponible dans les 2 modes débogage ou non-débogage...
}
#endif  //------------> si DEBUG n'est pas défini
#ifdef DEBUG
class MemCheck
{
        public:
                MemCheck(void *aptr, size_t amem_size, char fname[], int lineno);
                void    *ptr;
                size_t  mem_size;
                static  list<MemCheck>          mcH;  // tete de la liste
                static  unsigned long           total_memsize;  // memore totale allouee
};

// Variables globales
list<MemCheck>          MemCheck::mcH;
unsigned long           MemCheck::total_memsize = 0;
```

```
MemCheck::MemCheck(void *aptr, size_t amem_size, char fname[], int lineno)
{
        char func_name[100];
        FILE    *ferr = NULL;
        sprintf(func_name, "MemCheck() - File: %s Line: %d", fname, lineno);

        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(-1);
                #else
                        return;
                #endif
        }

        // Cherche si le pointeur existe déjà dans la liste...
        bool does_exist = false;
        list<MemCheck>::iterator iter1; // voir include/g++/stl_list.h
        //fprintf(ferr, "\n%s Before checking.. !!\n", func_name);
        list<MemCheck>::size_type sztype; // voir include/g++/stl_list.h

        // Le pointeur aptr n'existe pas dans la liste, alors on l'ajoute...
        //if (MemCheck::mcH.size() == 0)
        if (sztype == 0)
        {
                //fprintf(ferr, "\n%s aptr Not found\n", func_name);
                ptr = aptr;
                mem_size = amem_size;
                MemCheck::total_memsize += amem_size;

                // Voir aussi push_front(), list(), list(n), list(n, T)
                MemCheck tmpck = *this;
                MemCheck::mcH.push_back(tmpck);

                //MemCheck::mcH.insert(MemCheck::mcH.begin(), tmpck);
                //MemCheck::mcH.insert(MemCheck::mcH.end(), *this);
                fprintf(ferr, "\n%s sztype is %d\n", func_name, sztype);
                return;
        }
        if (MemCheck::mcH.empty() ) //if (MemCheck::mcH.empty() == true )
        {
                //fprintf(ferr, "\n%s List is empty!!\n", func_name);
        }
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
```

```
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
                        break;
                }
                if ( ((*iter1).ptr) == aptr)
                {
                        does_exist = true;
                        fprintf(ferr, "\n%s Already exists!!\n", func_name);
                        fprintf(ferr, "\n%s Fatal Error exiting now ....!!\n", func_name);
                        #ifdef DEBUG_MEM
                                exit(-1); //------------------------------------------------------------------>>>
                        #else
                                return;
                        #endif
                        // Change la taille de la mémoire pour de nouvelles valeurs
                        // Pour la taille totale - supprimer l'ancienne taille et ajouter la nouvelle
                        //fprintf(ferr, "\n%s total_memsize = %lu\n", func_name, (*iter1).total_memsize);
                        //fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                        //fprintf(ferr, "\n%s amem_size = %u\n", func_name, amem_size);
                        (*iter1).total_memsize = (*iter1).total_memsize + amem_size;
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize - (*iter1).mem_size;
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).mem_size);
                                        fprintf(ferr, "\n%s amem_size = %u\n", func_name, amem_size);
                                }
                        }
                        (*iter1).mem_size = amem_size;
                }
        }

        // Le pointeur aptr n'existe pas dans la liste, alors on l'ajoute maintenant
        if (does_exist == false)
        {
                //fprintf(ferr, "\n%s aptr Not found\n", func_name);
                ptr = aptr;
                mem_size = amem_size;
                MemCheck::total_memsize += amem_size;
                MemCheck::mcH.insert(MemCheck::mcH.end(), *this);
        }
        fclose(ferr);
}

static inline void call_check(void *aa, size_t tmpii, char fname[], int lineno)
{
```

```
        MemCheck bb(aa, tmpii, fname, lineno);
        if (& bb);  // une instruction inutile pour éviter un avertissement du compilateur
}

static inline void remove_ptr(void *aa, char fname[], int lineno)
{
        char    func_name[100];
        if (aa == NULL)
                return;

        sprintf(func_name, "remove_ptr() - File: %s Line: %d", fname, lineno);
        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(-1);
                #else
                        return;
                #endif
        }

        bool does_exist = false;
        if (MemCheck::mcH.empty() == true)
        {
                //fprintf(ferr, "\n%s List is empty!!\n", func_name);
                //fclose(ferr);
                //return;
        }
        list<MemCheck>::iterator iter1; // see file include/g++/stl_list.h
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
                        break;
                }
                if ( ((*iter1).ptr) == aa)
                {
                        does_exist = true;
                        // Change la taille de la mémoire pour les nouvelles valeurs
                        // Pour la taille totale - supprimer l'ancienne taille
                        //fprintf(ferr, "\n%s total_memsize = %lu\n", func_name, (*iter1).total_memsize);
                        //fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize - (*iter1).mem_size;
```

```
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                                }
                        }
                        MemCheck::mcH.erase(iter1);
                        break;  // break pour éviter une boucle infinie
                }
        }
        if (does_exist == false)
        {
                //fprintf(ferr, "\n%s Fatal Error: - You did not allocate memory!! \n", func_name);
                //fprintf(ferr, "\n%s The value passed is %s\n", func_name, (char *) aa);
        }
        else
                //fprintf(ferr, "\n%s found\n", func_name);
        fclose(ferr);
}

static inline void call_free_check(void *aa, char *fname, int lineno)
{
        char func_name[100];
        sprintf(func_name, "call_free_check() - File: %s Line: %d", fname, lineno);

        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(-1);
                #else
                        return;
                #endif
        }

        bool does_exist = false;
        list<MemCheck>::iterator iter1; // voir include/g++/stl_list.h
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
                        break;
                }
                if ( ((*iter1).ptr) == aa)
                {
```

```
                        does_exist = true;
                        //fprintf(ferr, "\n%s iter1.mem_size = %u\n", func_name, (*iter1).mem_size);
                        //fprintf(ferr, "\n%s Total memory allocated = %lu\n",  func_name, (*iter1).total_memsize);
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize - (*iter1).mem_size;
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).mem_size);
                                }
                        }
                        MemCheck::mcH.erase(iter1);
                        break;  // break pour éviter une boucle infinie
                }
        }
        if (does_exist == false)
        {
                fprintf(ferr, "\n%s Fatal Error: free() - You did not allocate memory!!\n",
                                func_name);
                //fprintf(ferr, "\n%s The value passed is %s\n", func_name, (char *) aa);
                fclose(ferr);
                #ifdef DEBUG_MEM
                        exit(-1);
                #else
                        return;
                #endif
        }
        else
        {
                //fprintf(ferr, "\n%s found\n", func_name);
        }
        fclose(ferr);
}

void local_print_total_memsize(char *fname, int lineno)
{
        char func_name[100];
        sprintf(func_name, "local_print_total_memsize() - %s Line: %d", fname, lineno);

        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(-1);
```

```
                #else
                        return;
                #endif
        }

        fprintf(ferr, "\n%s Total memory MemCheck::total_memsize = %lu\n",  func_name, MemCheck::total_memsize);
        fclose(ferr);
}
#endif  //------------> si DEBUG est défini
*/
```

18 Annexe E my_malloc.h

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//*****************************************************************
// La licence de distribution est la GNU/GPL et vous devez inclure
// le nom et le mel de l'auteur dans toutes les copies
// Auteur : Al Dev    Mel : alavoor@yahoo.com
//*****************************************************************

/*
**      Dans votre fonction main() mettre -
                char p_name[1024];
                sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
                putenv(p_name);
                print_total_memsize(); // au debut
                ......
                ......
                print_total_memsize(); // à la fin
*/

/* Utilisez zap au lieu de delete pour être très propre !
** Utilisez do while pour en faire une macro robuste et éviter les erreurs
*/
#define zap(x) do { if (x) { delete(x); x = 0; } } while (0)

void *local_my_malloc(size_t size, char fname[], int lineno);

char *local_my_realloc(char *aa, size_t size, char fname[], int lineno);
short *local_my_realloc(short *aa, size_t size, char fname[], int lineno);
void local_my_free(void *aa, char fname[], int lineno);

void local_print_total_memsize(char fname[], int lineno);

#define my_free(NM) (void) (local_my_free(NM, __FILE__, __LINE__))
#define my_malloc(SZ) (local_my_malloc(SZ, __FILE__, __LINE__))
#define my_realloc(NM, SZ) (local_my_realloc(NM, SZ, __FILE__, __LINE__))
```

```
#define print_total_memsize() (void) (local_print_total_memsize(__FILE__, __LINE__))

#ifdef DEBUG  //------------> DEBUG
#else  //------------> DEBUG
#define call_check(AA, BB, CC, DD) ((void) 0)
#define call_free_check(AA, BB, CC) ((void) 0)
#define remove_ptr(AA, CC, DD) ((void) 0)
#endif //------------> DEBUG
```

19 Annexe F debug.h

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//*****************************************************************
// La licence de distribution est la GNU/GPL et vous devez inclure
// le nom et le mel de l'auteur dans toutes les copies
// Auteur : Al Dev    Mel : alavoor@yahoo.com
//*****************************************************************
/****************************************************************
        Programme pour déboguer les programmes C++/C
*****************************************************************/

#define print_log(AA, BB, CC, DD, EE) ((void) 0)

#ifdef DEBUG

#include <iostream>
#include <string>
//#include <assert.h>  // macro assert() aussi utilisée pour déboguer

const bool LOG_YES = true;  // sortie envoyée dans le fichier de trace
const bool LOG_NO = false;  // pas de sortie dans le fichier de trace

// Code de débogage
// Utilisez debug2_ pour sortir le résultat dans le fichier de trace

#define debug_(NM, VL) (void) ( local_dbg(NM, VL, __FILE__, __LINE__) )
#define debug2_(NM, VL, LOG_FILE) (void) ( local_dbg(NM, VL, __FILE__, __LINE__, LOG_FILE) )

void local_dbg(char name[], char value[], char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], string value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], int value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], unsigned long value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], float value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], double value, char fname[], int lineno, bool logfile= false);

#else  //--------> sinon
```

```
#define debug_(NM, VL) ((void) 0)
#define debug2_(NM, VL, LOG_FILE) ((void) 0)

#endif // DEBUG
```

20 Annexe G debug.cpp

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//*****************************************************************
// La licence de distribution est la GNU/GPL et vous devez inclure
// le nom et le mel de l'auteur dans toutes les copies
// Auteur : Al Dev    Mel : alavoor@yahoo.com
//*****************************************************************

/****************************************************************
Programme pour déboguer des programmes C++/C
*****************************************************************/


#ifdef DEBUG  // fonctions nécessaires SEULEMENT si DEBUG est défini

#include "debug.h"
//#include "log.h"

// La variable value[] peut être de type char, string, int, unsigned long, float, etc.

void local_dbg(char name[], char value[], char fname[], int lineno, bool logfile) {
        if (value == NULL)
                return;
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %s\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], string value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %s\n", fname, lineno, name, value.c_str());
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value.c_str() << endl; }

void local_dbg(char name[], int value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], unsigned int value, char fname[], int lineno, bool logfile) {
```

```
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], long value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], unsigned long value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], short value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], unsigned short value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], float value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %f\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

void local_dbg(char name[], double value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %f\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " << value << endl; }

// Vous pouvez en ajouter plus ici - 'value' peut être une classe, une énumération, un champ date/heure, etc.


#endif // DEBUG
```

21 Annexe H Makefile

Vous pouvez récupérer tous les programmes en un seul tar.gz sur 2 (). Pour obtenir ce fichier, dans un butineur web, sauvez ce fichier en type 'texte'.

```
//*****************************************************************
// La licence de distribution est la GNU/GPL et vous devez inclure
// le nom et le mel de l'auteur dans toutes les copies
// Auteur : Al Dev    Mel : alavoor@yahoo.com
//*****************************************************************

.SUFFIXES: .pc .cpp .c .o

CC=gcc
CXX=g++

MAKEMAKE=mm
LIBRARY=libString.a
DEST=/home/myname/lib

# Pour construire la bibliothèque, et le programme de test principal, décommenter la ligne suivante :
#MYCFLAGS=-O -Wall

# Pour tester sans avoir trace de débogage, décommenter la ligne suivante :
#MYCFLAGS=-g3 -Wall

# Pour permettre un 'débogage complet', décommenter la ligne suivante :
MYCFLAGS=-g3 -DDEBUG -Wall

#PURIFY=purify -best-effort

SRCS=my_malloc.cpp String.cpp StringTokenizer.cpp debug.cpp example_String.cpp
HDR=my_malloc.h  String.h StringTokenizer.h debug.h
OBJS=my_malloc.o String.o StringTokenizer.o debug.o example_String.o
EXE=String

# Pour générer les dépendances du Makefile
SHELL=/bin/sh

CPPFLAGS=$(MYCFLAGS) $(OS_DEFINES)
CFLAGS=$(MYCFLAGS) $(OS_DEFINES)

#
# Si libString.a se trouve dans le répertoire
# courant, utilisez -L. (tiret L point)
MYLIBDIR=-L$(MY_DIR)/libmy -L.

ALLLDFLAGS= $(LDFLAGS)  $(MYLIBDIR)

COMMONLIBS=-lstdc++ -lm
MYLIBS=-lString
LIBS=$(COMMONLIBS)  $(MYLIBS)

all: $(LIBRARY) $(EXE)
```

```
$(MAKEMAKE):
	@rm -f $(MAKEMAKE)
	$(PURIFY) $(CXX) -M  $(INCLUDE) $(CPPFLAGS) *.cpp > $(MAKEMAKE)

$(EXE): $(OBJS)
	@echo "Creating a executable "
	$(PURIFY) $(CC) -o $(EXE) $(OBJS) $(ALLLDFLAGS) $(LIBS)

$(LIBRARY): $(OBJS)
	@echo "\n***********************************************"
	@echo "   Loading $(LIBRARY) ... to $(DEST)"
	@echo "**********************************************"
	@ar cru $(LIBRARY) $(OBJS)
	@echo "\n "

.cpp.o: $(SRCS) $(HDR)
#	@echo "Creating a object files from " $*.cpp " files "
	$(PURIFY) $(CXX) -c  $(INCLUDE) $(CPPFLAGS) $*.cpp

.c.o: $(SRCS) $(HDR)
#	@echo "Creating a object files from " $*.c " files "
	$(PURIFY) $(CC) -c $(INCLUDE) $(CFLAGS) $*.c

clean:
	rm -f *.o *.log *~ *.log.old *.pid core err a.out lib*.a afiedt.buf
	rm -f $(EXE)
	rm -f $(MAKEMAKE)

#%.d: %.c
#	@echo "Generating the dependency file *.d from *.c"
#	$(SHELL) -ec '$(CC) -M $(CPPFLAGS) $< | sed '\''s/$*.o/& $@/g'\'' > $@'
#%.d: %.cpp
#	@echo "Generating the dependency file *.d from *.cpp"
#	$(SHELL) -ec '$(CC) -M $(CPPFLAGS) $< | sed '\''s/$*.o/& $@/g'\'' > $@'

# Doit inclure tous les flags C pour l'option -M
#$(MAKEMAKE):
#	@echo "Generating the dependency file *.d from *.cpp"
#	$(CXX) -M  $(INCLUDE) $(CPPFLAGS) *.cpp > $(MAKEMAKE)

include $(MAKEMAKE)
#include $(SRCS:.cpp=.d)
#include $(SRCS:.c=.d)
```