```cpp
/*************************************************************************
Log  -  description
-------------------
début                : 23/11/2015
copyright            : (C) 2015 par mfallouh_mvirsolvy
*************************************************************************/

//---------- Réalisation de la classe <Log> (fichier Log.cpp) --

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>

//------------------------------------------------------ Include personnel
#include "Log.h"

//------------------------------------------------------------------ Constantes

//---------------------------------------------------- Variables de classe

//-------------------------------------------------------------- Types privés


//----------------------------------------------------------------- PUBLIC
//---------------------------------------------------------- Fonctions amies

//---------------------------------------------------------- Méthodes publiques

//------------------------------------------------------ Surcharge d'opérateurs

/*
Log & Log::operator = ( const Log & unLog )
// Algorithme :
//
{
} //----- Fin de operator =
*/

//-------------------------------------------- Constructeurs - destructeur


Log::Log ( const Log & unLog )
// Algorithme :
//
{
#ifdef MAP
cout << "Appel au constructeur de copie de <Log>" << endl;
#endif
       ip = unLog.ip;
       logname = unLog.logname;
       user = unLog.user;
       date = unLog.date;
       statut = unLog.statut;
       taille = unLog.taille;
       referer = unLog.referer;
       navigateur = unLog.navigateur;
```

```
} //----- Fin de Log (constructeur de copie)


Log::Log(const string &log)
// Algorithme :
// Parcours la string log, en s'arretant sur des séparateurs particuliers.
{
#ifdef MAP
	cout << "Appel au constructeur de <Log>" << endl;
#endif
	if (log.empty()) // chaine vide
	{
		cerr << "[LOG] le log est vide " << endl;
		setErreur();
		return;
	}
	else
	{
		size_t posDebut, posFin; // variables de position pour parcourir la
chaine

		posDebut = 0;
		posFin = 0;

		/*récupération de l'ip*/
		if ((posFin = log.find(SEP,posDebut)) != log.npos)		// SEP
trouvé
		{
			ip = log.substr(posDebut, posFin);
			posDebut = posFin +1;
		}
		else
	// SEP non trouvé
		{
			cerr << "[LOG] erreur dans <ip> : " << log << endl;
			setErreur();
			return;
		}

		/*récupération user*/
		if ((posFin = log.find(SEP,posDebut)) != log.npos)		// SEP
trouvé
		{
			user = log.substr(posDebut,posFin-posDebut);
			posDebut = posFin + 1;
		}
		else
	// SEP non trouvé
		{
			cerr << "[LOG] erreur dans <user> : " << log << endl;
			setErreur();
			return;
		}

		/*récupération logname*/
		if ((posFin = log.find(SEP,posDebut)) != log.npos)		// SEP
trouvé
		{
```

```
                        logname = log.substr(posDebut, posFin - posDebut);
                        posDebut = posFin + 2;
                }
                else
        // SEP non trouvé
                {
                        cerr << "[LOG] erreur dans <logname> : " << log << endl;
                        setErreur();
                        return;
                }

                /*récupération date*/
                if ((posFin = log.find(SEP_DATE_FIN,posDebut)) != log.npos)//
SEP_DATE_FIN trouvé
                {
                        date = log.substr(posDebut, posFin - posDebut);
                        posDebut = posFin + 3;

        //on suprime bien SEP_DATE_FIN et SEP qui le suit
                }
                else
        // SEP_DATE_FIN non trouvé
                {
                        cerr << "[LOG] erreur dans <date> : " << log << endl;
                        setErreur();
                        return;
                }

                /*récupération statut*/
                if ((posFin = log.find(SEP, posDebut)) != log.npos)          // SEP
trouvé
                {
                        statut = log.substr(posDebut, posFin - posDebut);
                        posDebut = posFin + 1;
                }
                else
        // SEP non trouvé
                {
                        cerr << "[LOG] erreur dans <statut> : " << log << endl;
                        setErreur();
                        return;
                }


                /*on enleve la partie requete*/
                if ((posFin = log.find(SEP_REQ,posDebut)) != log.npos)// SEP_REQ
trouvé
                {
                        posDebut = posFin + 2; //on supprime la partie concernant la
requete
                }
                else
        // SEP_REQ non trouvé
                {
                        cerr << "[LOG] erreur dans <requête> : " << log << endl;
                        setErreur();
                        return;
                }
```

```
		/*on enleve la partie retour */
		if ((posFin = log.find(SEP, posDebut)) != log.npos)	// SEP_REQ
trouvé
		{
			posDebut = posFin + 1;
		}
		else
	// SEP_REQ non trouvé
		{
			cerr << "[LOG] erreur dans <retour> : " << log << endl;
			setErreur();
			return;
		}
		/*récupération taille*/
		if ((posFin = log.find(SEP,posDebut)) != log.npos)		// SEP
trouvé
		{
			taille = log.substr(posDebut, posFin - posDebut);
			posDebut = posFin + 2;
		}
		else
	// SEP non trouvé
		{
			cerr << "[LOG] erreur dans <taille> : " << log << endl;
			setErreur();
			return;
		}

		/*récupération url*/
		if ((posFin = log.find(SEP_REQ, posDebut)) != log.npos)
	//SEP_REQ trouvé
		{
			url = log.substr(posDebut, posFin - posDebut);
			// on ne modifie pas la position de début
		}
		else
	// SEP_REQ non trouvé
		{
			cerr << "[LOG] erreur dans <url> : " << log << endl;
			setErreur();
			return;
		}

		referer = url;
		if (referer.find(URL_LOCALE) != string::npos)
		{
			referer.erase(0, URL_LOCALE.size());
		}
		referer = referer.substr(0, referer.find(SEP_PVIRG)); //enlever tout
ce qui est apres un ;
		referer = referer.substr(0, referer.find(SEP_INT)); //enlever tout ce
qui est après un ?
		referer = referer.substr(0, referer.find(SEP_PARAM)); //enlever tout
ce qui est après un /&

		/* récupération navigateur*/
		posDebut = log.find(SEP_REQ, posFin) + 1;
```

```
		posDebut = log.find(SEP_REQ, posDebut) + 1;
		if ((posFin = log.find(SEP_REQ, posDebut)) != string::npos)  // SEP
trouvé
		{
			navigateur = log.substr(posDebut, posFin - posDebut);
			posDebut = posFin + 1;
		}
		else
	// SEP non trouvé
		{
			cerr << "[LOG] erreur dans <navigateur> : " << log << endl;
			setErreur();
			return;
		}


		if (posDebut!=log.size())  // il n'y a pas plus d'infos dans le log.
		{
			cerr << "[LOG] erreur de remplissage, il reste: ";
			cerr << log.substr(posDebut,log.npos) << endl;
		}
	} // fin du traitement sans erreur

#ifdef MAP
	cout << "Les valeurs du Log sont :" << endl;
	  cout << "ip"<<ip << endl;
	  cout <<"logname"<< logname << endl;
	  cout <<"user"<< user << endl;
	  cout << "date"<<date << endl;
	  cout << "statut"<<statut << endl;
	  cout << "taille"<<taille << endl;
	  cout << "URL" << url << endl;
	  cout << "referer"<<referer << endl;
	  cout <<"nvigateur"<< navigateur << endl;
#endif
} //----- Fin de Log


Log::~Log()
// Algorithme :
//
{
#ifdef MAP
	cout << "Appel au destructeur de <Log>" << endl;
#endif
} //----- Fin de ~Log

  //------------------------------------------------------------------ PRIVE

  //----------------------------------------------------- Méthodes protégées

  //------------------------------------------------------- Méthodes privées
void Log::setErreur()
// Algorithme :
//
{
	ip = ERREUR;
	logname = ERREUR;
```

```
        user = ERREUR;
        date = ERREUR;
        statut = ERREUR;
        taille = ERREUR;
        referer = ERREUR;
        url = ERREUR;
        navigateur = ERREUR;
    }// ----- Fin de setErreur
```