TP 1
INDEXATION DE CODE

Introduction

Pour le deuxième TP, vous devez construire un analyseur de code « C++ » qui cherche les identificateurs déclarés dans un programme. Pour chaque identificateur différent trouvé, le programme conserve le numéro des lignes où il est rencontré. Pour conserver cette information, nous allons utiliser un arbre AVL. Vous devrez construire cette structure d'arbre.

Description

Identificateurs

Un identificateur C++ est une suite de caractères qui commence par une *lettre* ('a'..'z', 'A'..'Z' ou '_') et qui est suivie par une suite de *lettres* et/ou de *chiffres* ('0'..'9'). Vous devez donc parcourir le code pour trouver ces suites. Les mots-clés du langage ne sont pas considérés comme des identificateurs.

Mots clés de type directive :

define	endif	ifdef	line
elif	error	ifndef	pragma
else	if	include	undef

Mots clés de base :

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

Mots clés pour les représentations alternatives :

and	bitor	not_eq	xor
and_eq	compl	or	xor_eq
bitand	not	or_eq	

Vous devrez aussi ignorer le texte qui est en commentaire et dans les chaînes de caractères. Aussi, lorsque vous trouvez la directive `include`, vous devez ignorer le texte entre les '<' et '>' qui la suive.

La première ligne du fichier en entrées est numérotée : 1. Chaque saut de ligne ajoute '1' au numéro de ligne. Bien que les mots dans les commentaires ne soient pas retenus, les fins de lignes entre commentaires donnent un incrément au numéro de ligne. Important : si la directive #ligne est rencontrée, et qu'elle est suivie par un numéro, alors se numéro devient le numéro de la ligne suivante dans le code.

Utilisation

Votre programme doit être utilisable à partir d'une ligne de commande Linux. Cette commande va recevoir un paramètre qui indique le nom du fichier contenant le programme à traiter. Un second paramètre, **optionnel**, indiquera le nom du fichier où les résultats seront écrits. Si ce second paramètre est omis, alors le résultat devra être affiché sur la sortie standard (`cout`).

Chaque ligne de la sortie aura le format suivant, à raison d'un (1) identificateur par ligne :

ident : *liste_no_ligne*

Les différents éléments de chaque ligne ont la signification suivante :

- *ident* est l'identificateur trouvé.
- *liste_no_ligne* est la suite des numéros de ligne où l'identificateur a été rencontré.

Par exemple soit le programme suivant :

```
#include <iostream>

int add( int x, int y) {
  return x + y;
}

int main()  {
  cout << add( 5, 4 );
  return 0;
}
```

Alors la sortie suivante sera produite :

```
add : 3, 8
x : 3, 4
```

```
y : 3, 4
main : 7
cout : 8
```

Tests

Votre application doit contenir les tests nécessaires pour démontrer son bon fonctionnement. Ce sera votre responsabilité de prouver le bon fonctionnement, donc de construire les tests nécessaires pour couvrir les utilisations de votre logiciel.

Directive

1. Le tp est à faire seul ou en équipe de deux (maximum).
2. Commentaire :
    a. Commentez l'entête de chaque fonction. Ces commentaires doivent contenir la description de la fonction et le rôle de ces paramètres.
    b. Une ligne contient soit un commentaire, soit du code, pas les deux.
    c. Utilisez des noms d'identificateur significatif.
    d. Utilisez le français.
3. Code :
    a. Pas de `goto`, `continue`.
    b. Les `break` ne peuvent apparaître que dans les `switch`.
    c. Un seul `return` par méthode.
4. Indentez votre code. Assurez-vous que l'indentation est faite avec des espaces.

Remise

Vous devez remettre votre code (votre programme principal, vos modules, le module {\tt ArbreAVL}, les tests et un **makefile**. Le tout doit être archivé (.tar) et ensuite compressé (.gz) sur Malt. Remettre le tp par l'entremise de Moodle. Remettez votre fichier `'*.tar.gz'`. Le tp est à remettre avant le 7 juillet 23:59.

Évaluation

- Fonctionnalité (14 pts) : Votre tp doit compiler sans erreur (il peut y avoir des warnings). J'utilise la ligne de compilation suivante : `make`
- Structure (3 pt) : Il faut avoir **plusieurs** fonctions et des classes. Construisez un code bien structuré.
- Lisibilité (3 pts) : commentaire, indentation et noms d'identificateur significatif.