Sujet Complémentaire 1 : La chaîne de compilation

La chaîne de compilation

Lorsque vous appelez en ligne de commande le « compilateur » `g++` vous déclenchez en fait une chaîne de compilation qui fait intervenir 4 logiciels en 4 étapes, chacune de ces étapes produisant un ou plusieurs fichiers spécifiques et éventuellement des erreurs :

1. *le préprocesseur* réunit tous les fichiers sources en un seul fichier texte ;
2. *le compilateur* transforme cette source en code assembleur ;
3. *l'assembleur* transforme ce fichier en langage machine (fichier binaire) ;
4. *le lieur* produit un exécutable avec tous les fichiers binaires impliqués, y compris les bibliothèques. En général c'est cet exécutable qui nous intéresse et qui est le seul fichier conservé.

Étape 1

Dans un fichier nommé `programmme.cc` copiez le code suivant (y compris le commentaire).

```
int main()
{
    // commentaire
    int a;
    a = 123;
    return 0;
}
```

Compilez ce programme avec `g++` en utilisant l'option `-save-temps`. Affichez le contenu du répertoire.

Q1. Qu'observez-vous ? Identifiez, en affichant leur contenu, à quelle étape chaque fichier a été créé.

Q2. Que font les options `-E`, `-S`, `-c` de `g++` ? Note : l'option `--help` peut vous aider... Testez chacune des options (sans oublier d'effacer les fichiers produits avant chaque test).

Q3. Quelles sont les différences entre le fichier source et le fichier produit par le préprocesseur ? Expliquez l'absence du commentaire, la présence de lignes vides.

Étape 2

Transformez le programme précédent de la façon suivante.

```
int main()
{
    // commentaire
    int a;
    a = V1;
    return 0;
}
```

Q4. Ce programme est-il correct ? Que va-t-il se produire à la compilation ?

Après avoir effacé les fichiers produits par l'étape précédente, compilez ce programme avec l'option `-save-temps`. Comme vous avez dû le prévoir, une erreur de compilation se produit.

Q5. Observez quels sont les fichiers écrits et quelles sont leurs différences avec la précédente étape. Expliquez.

Étape 3

Créez un fichier `variables.h` avec le contenu suivant :

```
int V1 = 123;
double V2 = 5.2;
```

Ajoutez une directive `#include` au fichier `programme.cc` pour inclure ce fichier.

Q6. Ce programme est-il correct à présent ? Compilez et observez le contenu des fichiers produits comme pour l'étape 1, spécialement le fichier produit par le préprocesseur.

Étape 4

Doublez la directive `#include` dans le fichier `programme.cc`.

```
#include "variables.h"
#include "variables.h"
```

Q7. Le programme est-il toujours correct ? Que va-t-il se produire à la compilation ?

Observez le fichier produit par le préprocesseur et faites le rapprochement avec le message d'erreur du compilateur.

Sans modifier le fichier `programme.cc` utilisez des directives de précompilation `#ifndef` et `#define` dans `variables.h` pour remédier à ce problème.

```
#ifndef __VARIABLES_H__
#define __VARIABLES_H__
int V1 = 123;
double V2 = 5.2;
#endif
```

Q8. À quoi servent ces directives ? Compilez, observez fichier produit par le préprocesseur et expliquez.

Étape 5

À titre de curiosité, observez la taille et le contenu des fichiers produits par la compilation de `programme.cc` après y avoir inclus (par exemple) `iostream` par une directive `#include`, toujours avec l'option `-save-temps` bien évidemment.

Q9. Que pouvez-vous en conclure sur les inclusions inutiles ?