Note : Codes sources disponibles uniquement en C.

Décomposer pour simplifier

Les fonctions sont un outil incontournable pour produire des programmes qui peuvent être repris et étendus. Elles permettent d'une part d'éviter les répétitions de blocs d'instructions similaires, et d'autre part de décomposer le programme en éléments simples.

Pour résoudre un problème très complexe, un moyen efficace consiste à le décomposer en sous-problèmes avant de résoudre chacun des sous-problèmes indépendamment. Chaque sous-problème est alors plus simple à appréhender que le problème complet. On peut ensuite décomposer à leur tour les sous-problèmes obtenus, jusqu'à obtenir des problèmes simples à résoudre. Décomposer un problème en sous-problèmes n'est pas toujours une tâche facile, mais en le faisant, on réduit considérablement la difficulté de sa résolution.

Les fonctions permettent d'appliquer cette technique à la réalisation de programmes. Prenons par exemple le programme suivant :

```c
#include <stdio.h>

int main()
{
   int nbColonnes, nbLignes;

   scanf("%d", &nbColonnes);
   for (int iCol = 0; iCol < nbColonnes; iCol = iCol + 1)
   {
      printf("X");
   }
   printf("\n");

   scanf("%d", &nbLignes);
   for (int iLigne = 0; iLigne < nbLignes; iLigne = iLigne + 1)
   {
      for (iCol = 0; iCol < iLigne + 1; iCol = iCol + 1)
      {
         printf("*");
      }
      printf("\n");
   }
   printf("\n");

   scanf("%d%d", &nbLignes, &nbColonnes);
   for (int iLigne = 0; iLigne < nbLignes; iLigne = iLigne + 1)
   {
      for (int iCol = 0; iCol < nbColonnes; iCol = iCol + 1)
      {
         printf("#");
      }
      printf("\n");
   }
}
```

Cet exemple montre qu'un source plutôt bien présenté, avec des noms de variables choisis correctement et qui réalise quelque chose de pourtant très simple, peut malgré tout devenir assez indigeste. On peut bien sûr le comprendre en le lisant entièrement ligne par ligne, mais il faut un peu de temps pour comprendre ce qu'il fait exactement. Cela reste raisonnable, car nous

n'allons pas vous imposer la lecture du source d'un véritable projet, mais cela ne fait que s'aggraver avec la taille du programme si l'on ne prend pas le soin d'utiliser les fonctions.

Regardez maintenant le source suivant, qui fait exactement la même chose que l'exemple précédent, mais est cette fois organisé en fonctions :

```
#include <stdio.h>

void afficheLigne(int nbColonnes, char motif)
{
   for (int iCol = 0; iCol < nbColonnes; iCol = iCol + 1)
   {
      printf("%c", motif);
   }
   printf("\n");
}

void afficheTriangle(int nbLignes, char motif)
{
   for (int iLigne = 0; iLigne < nbLignes; iLigne = iLigne + 1)
   {
      int nbColonnes = iLigne + 1;
      afficheLigne(nbColonnes, motif);
   }
}

void afficheRectangle(int nbLignes, int nbColonnes, char motif)
{
   for (int iLigne = 0; iLigne < nbLignes; iLigne = iLigne + 1)
      afficheLigne(nbColonnes, motif);
}

int main()
{
   int nbColonnes, nbLignes;

   scanf("%d", &nbColonnes);
   afficheLigne(nbColonnes, 'X');

   scanf("%d", &nbLignes);
   afficheTriangle(nbLignes, '*');

   scanf("%d%d", &nbLignes, &nbColonnes);
   afficheRectangle(nbLignes, nbColonnes, '#');
}
```

Cette nouvelle version du même programme est bien plus facile à comprendre et bien plus agréable à lire, ceci pour plusieurs raisons.

- Une fonction a été créée pour chacune des trois choses que fait le programme : afficher une ligne, un triangle et un rectangle. Chaque partie peut donc être lue séparément, ce qui est plus facile que de la lire au milieu d'autres instructions.
- Le nom donné à chaque fonction décrit clairement ce qu'elle fait. Le nom des paramètres permet également de comprendre tout de suite à quoi ils correspondent. On peut ainsi lire les instructions de chaque fonction en sachant déjà ce que ces instructions sont supposées faire, ce qui aide beaucoup.
- La fonction principale *main* ne contient quasiment plus que des appels à ces fonctions et est un résumé qui décrit très clairement, en quelques lignes, ce que fait l'ensemble du programme. Les noms des fonctions bien choisis font que ces lignes se comprennent tout de suite.

- Les instructions d'affichage d'une ligne de symboles, utilisées dans les trois parties du programme, ne sont plus recopiées à chaque fois mais écrites une fois pour toutes dans la fonction *afficheLigne*, qui est ainsi appelée par les deux autres fonctions.

Pour obtenir un source facile à lire et à comprendre, nous vous conseillons donc d'appliquer les mêmes principes à vos programmes :

1. consacrer une fonction à chaque action distincte ;
2. choisir des noms de fonctions qui décrivent ce qu'elles font ;
3. regrouper les parties de code similaires en fonctions, pour éviter les répétitions ;
4. ne pas écrire de fonctions trop longues, éviter de dépasser 25 lignes.

Appliquer systématiquement ces conseils peut parfois sembler ennuyeux, surtout lorsque l'on est en plein dans l'écriture de code, que l'on se sent inspiré et qu'on a envie d'avancer rapidement. Le temps que l'on économise en ne prenant pas la peine d'appliquer ces divers conseils finit cependant toujours par être largement reperdu plus tard, en recherche de bogues ou modifications. L'effort consistant à bien organiser son code est en outre bien plus gratifiant que celui passé à chercher des bogues dans un source incompréhensible.

Retour sur les commentaires

On notera que dans la plupart des langages, contrairement aux blocs d'instructions, les commentaires ne peuvent pas s'imbriquer les uns dans les autres. Par exemple, ceci est invalide et ne compilera pas :

```
/*
   devenu inutile :
   printf("\n"); /* On passe à la ligne */
*/
```

On peut en revanche utiliser la compilation conditionnelle (les directives de préprocesseur) pour ignorer facilement une partie du programme :

```
#if 0
   printf("Instruction en attente\n");
#endif
```

qui sont alors imbricables. Le code dans le `#if 0` sera complètement ignoré à la compilation. Dans les langages ne proposant pas la compilation conditionnelle, on peut utiliser des `if` normaux pour sauter une partie du programme, bien qu'il faille alors que le code à l'intérieur soit valide.

Ajouter des commentaires à votre source est un bon moyen pour le rendre plus facilement lisible par d'autres, ou par vous-même plus tard. Avoir mis des commentaires ne doit cependant pas être une excuse pour avoir par ailleurs un code difficile à lire, des noms de variables et de fonctions peu clairs, etc. Avant d'ajouter des commentaires, réfléchissez à la manière dont vous pouvez modifier votre source pour qu'il soit compréhensible de lui-même. Après seulement, envisagez l'ajout de commentaires pour expliquer ce qui n'est pas évident à la lecture du code (et ne peut être rendu évident simplement).

Évitez autant que possible de mettre des commentaires au milieu du code. En essayant d'expliquer ce que fait le source, ils ont l'inconvénient de le rendre plus long et moins facile à lire. Préférez des commentaires à l'extérieur des fonctions. Si votre fonction fait beaucoup de choses et que vous avez envie de mettre des commentaires entre les différentes parties pour expliquer ce qu'elles font, c'est probablement que votre fonction mérite d'être décomposée en sous-

fonctions. Le code suivant montre comme les commentaires manquent d'intérêt et peuvent ralentir la lecture :

```c
#include <stdio.h>

// La fonction qui suit permet d'afficher une ligne de caractères.
// Elle prend en paramètre le nombre de caractères à afficher et
// le caractère à afficher.

void afficheLigne(int nbColonnes, char motif)
{
   int iCol = 0;
   // Tant qu'on n'a pas atteint le nombre de colonnes indiqué
   while (iCol < nbColonnes)
   {
      printf("%c", motif); // On affiche le caractère
      iCol = iCol + 1; // On passe à la colonne suivante
   }
   printf("\n"); // On passe à la ligne
}
// La fonction est terminée
```

Les commentaires peuvent être très utiles en cours de développement, en particulier pour les fonctions qui ne sont pas terminées. Par exemple, vous pouvez insérer des commentaires aux endroits où il reste des choses à faire, à améliorer, ou à modifier. Plutôt que de remettre simplement quelque chose à plus tard, au risque de l'oublier, mettez une note en commentaire à l'endroit concerné. Une notation usuelle pour indiquer une chose restant à faire consiste à commencer le commentaire par le texte « TODO: » (qui vient de l'anglais ‘to do’ : « à faire ») :

```c
int main()
{
   int nbColonnes;
   scanf("%d", &nbColonnes);
   // TODO: vérifier que l'on n'entre pas une valeur négative ou trop grande
   afficheLigne(nbColonnes, 'X');
}
```

Une autre utilisation consiste à écrire d'abord en français le pseudo-code d'une fonction, en le laissant en commentaire, pour que le programme compile toujours, en attendant de l'avoir écrite en C.