Introduction à l'algorithmique et à la programmation
Programmation modulaire en C

La programmation modulaire

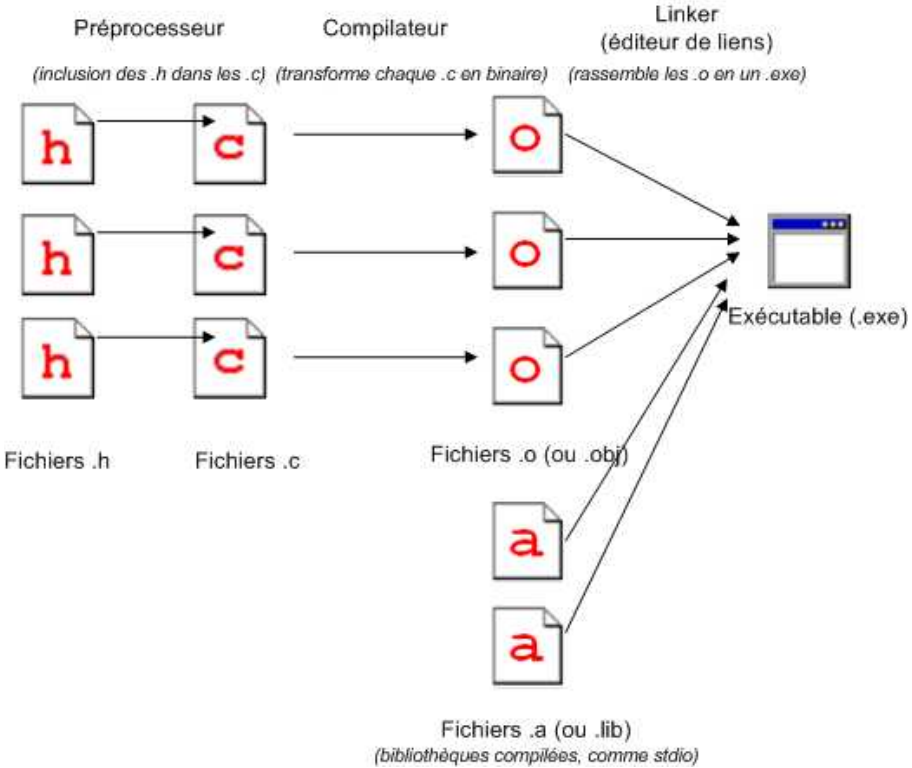
Présentation :

- Lorsque les programmes deviennent conséquents, il devient nécessaire de travailler avec plusieurs fichiers.
- Lorsque les programmes deviennent conséquents, plusieurs équipes vont travailler en parallèle (une équipe graphique, une équipe réseaux, ...).
- Des programmes peuvent intégrer des fonctionnalités développées par des entreprises extérieures (… qui ne souhaitent pas transmettre les codes sources).

- La programmation modulaire permet de réutiliser plus facilement les modules développés antérieurement.

La programmation modulaire

Rappel concernant la compilation :



La compilation : le préprocesseur

- Le préprocesseurs :

  - Le **préprocesseur** est un programme qui réalise des traitements sur le code source ***avant*** la compilation proprement dite.

  - Les instructions destinées au préprocesseur commence par #

```
# include <stdio.h>     // copie toutes les lignes du fichier stdio.h
# define MAX 100        // remplace MAX par 100 dans le code du fichier concerné
```

- Pour visualiser le résultat produit par le préprocesseur, on utilise l'option -E de gcc

```
gcc -o prog.txt -E prog.c
```

La compilation proprement dite

O Le compilateur va générer des fichiers objet (fich.o) propre à chaque machine (dépendant du système d'exploitation)

- Deux types de problèmes peuvent être détectés :

  *a) **Les erreurs de compilation** surviennent quand le compilateur ne peut pas produire le code machine associé à votre programme (erreurs de syntaxe, erreurs de typage, ...).*
  *b) **Des avertissements** (warnings en anglais) peuvent être affichés quand des incohérences sont repérées, mais que la génération d'un exécutable peut quand même aboutir.*

- Pour visualiser le résultat produit par le compilateur, on utilise l'option -c de gcc :

```
gcc -c prog.c
```

- le compilateur va générer le fichier **prog.o (compilé donc illisible)**

O Arrêt après l'étape de génération du code assembleur :

- Il est possible de visualiser le code assembleur du programme :

```
gcc -S prog.c
```

- Le compilateur va généré le fichier **prog.s (fichier texte)**

- Il est possible de relancer la compilation à partir du fichier prog.s

```
gcc prog.o  ou gcc prog.s ou gcc prog .c
```

La compilation : synthèse

Commande gcc	Action et commentaire
gcc -o prog.txt -E prog.c	Visualisation du fichier après passage du pré-processeur
gcc -S prog.c	Arrêt de la compilation au code assembleur génération du fichier prog.s
gcc -c prog.c	Fin de la compilation Pas d'édition de lien génération du fichier prog.o

La programmation modulaire

L’édition de lien :

- L’édition de lien est la dernière phase de création d’un fichier exécutable.
- Il est possible de lancer d’édition de lien à partir de fichiers de différents formats:
  - Des fichiers sources : .c
  - Des fichiers objets : .o
  - Des bibliothèques compilées

```
gcc -o prog.out prog.c fonctions.o ...
```

- L’édition de lien produit un fichier exécutable

La programmation modulaire

Rappel : fichier unique : la bonne organisation d'un programme :

```
// bibliothèques standards
# include <.stdio.h>
// constantes symboliques
# define LONG 30
// déclarations des types et structures
typedef struct{
        char c_nom [LONG];
} t_personne;
// définitions des constantes
const t_personne PERS_VIDE ={"sans_nom"};
// définitions des prototypes des fonctions
void init(t_personnel *adrPers);
// programme principal
int main(){
        ...
        return 0 ;
}
// définitions des fonctions
void init(t_personnel *adrPers){
        ...
}
```

La programmation modulaire

```
// bibliothèque standard
# include <.stdio.h>
```

const.h

```
// constantes symboliques
# define LONG 30
```

types.h

```
// déclarations des types et structures
typedef struct{
        char c_nom [LONG];
} t_personne;
```

globales.h et globales.c

```
// définitions des constantes / variables globales
const t_personne PERS_VIDE ={"sans_nom"};
```

fonction.h

```
// définitions des prototypes des fonctions
void init(t_personnel *adrPers);
```

main.c

```
// programme principal
int main(){
        ...
        return 0 ;
}
```

fonction.c

```
// définitions des fonctions
void init(t_personnel *adrPers){
         strcpy(adrPers->c_nom,"sans_nom");
}
```

La programmation modulaire

Les fichiers .h

- Protection contre les inclusions multiples (fichier.h)

```
# ifndef NOM_FICHIER_H      // if not define
# define NOM_FICHIER_H      // définition d'une constante symbolique spécifique
....
# endif                     // end if ... associée à #ifndef
```

- const.h (les constantes symboliques)

```
# ifndef CONST_H // protection contre les inclusions multiples
# define CONST_H
// constantes symboliques
# define LONG 30
# endif
```

- types.h

```
# ifndef TYPES_H
# define TYPES_H
#include "const.h"
// types et structures
typedef struct{
        char c_nom [LONG];
} t_personne;
# endif
```

La programmation modulaire

- globales.h

```
# ifndef GLOBALES_H
# define GLOBALES_H
# include "types.h"
// constantes : variables globales
extern const t_personne PERS_VIDE;      // extern : la variable sera définie dans un autre
                                        // fichier
# endif
```

- mot clé extern => pas de réservation mémoire
- Le fichier globales.h indispensables pour les fichiers de définitions (.c)

- globales.c

```
# include "types.h"
# include "globales.h" // vérification de la conformité avec globales.h
// definition des constantes
const t_personne PERS_VIDE ={"sans_nom"};
```

La programmation modulaire

Les fichiers .h et .c

- fonctions.h (les prototypes des fonctions)

```
# ifndef FONCTIONS_H
# define FONCTIONS_H
# include "types.h"
// définition des prototypes
void init(t_personne *adrPers);
t_personne saisir_personne();
void affiche(t_personne pers);
# endif
```

- fonction.c (définitions des fonctions)

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include "types.h"
# include "globales.h"
# include "fonctions.h" // vérification de la conformité entre les prototypes et les entêtes des fonctions
// definition des fonctions
void init(t_personne *adrPers){...}
t_personne saisir_personne(){...}
void affiche(t_personne pers){...}
```

La programmation modulaire

Le fichier du main :

O main_personne.c

```
# include <stdio.h>
// constantes symboliques
# include "const.h"
// types et structures
# include "types.h"
// définition des prototypes
# include "fonctions.h"
// définition des constantes
# include "globale.h"
// programme principal
int main(){
        ...
        return 0 ;
}
```

La programmation modulaire

La compilation séparée :

- Génération des fichiers objet (.o)

```
gcc -Wall -c globales.c          // génération du fichier globales.o
gcc -Wall -c fonctions.c         // génération du fichier fonction.o
gcc -Wall -c main_personne.c     // génération du fichier  main_gestion_personnel.o
```

L'édition de lien : génération du fichier exécutable

```
gcc main_gestion_personnel.o fonctions.o globales.o -o gestion_personne.out
```

Lancement du programme exécutable :

```
./ main_gestion_personnel.out
```

… rappel :

- fichier.h => inclusion avec **#include**
- fichier.c ou fichier.o => compilation avec **gcc**

La programmation modulaire

… un script pour exécuter toutes les instructions

- Fichier make.sh

```
#!/bin/bash
rm *.o              # supprimer les fichiers .o
rm *.out            # supprimer les fichiers .out

# compilation séparée
gcc -Wall -c globales.c
gcc -Wall -c fonctions.c
gcc -Wall -c main_personne.c

# edition de lien
gcc -Wall main_personne.o fonctions.o globales.o -o gestion_personne.out

# lancement du programme ... si tout va bien
./gestion_personne.out
```

- pour l'exécution :

```
sh make.sh
```