Modules & librairies en C/C++

- Programmation modulaire
  - motivations :
    - 1 programme <=> 1 fichier source ?

Modules & librairies en C/C++

- Programmation modulaire
  - motivations :
    - 1 programme <=> 1 fichier source ?
      - 1 seul développeur actif !!!
      - réutilisabilité nulle
        - impossible sans recompilation
        - possible si on dispose des sources mais difficile
      - inefficacité
        - recompilation systématique de tout le code
        - l'ensemble des sources dans l'éditeur

Modules & librairies en C/C++

- Programmation modulaire
  - motivations :
    - **correction et maintenance difficiles**
      - manipulation constante de toutes les sources
      - risque d'erreur involontaire
      - risque de manipulation volontaire inadéquate
      - mauvaise localisation des erreurs
      - interdépendance de toute les parties du code
      - ...

Modules & librairies en C/C++

- Programmation modulaire
  - principes :
    - module :
      - structure les sources
        - **1 module = 1 fichier d'implémentation** (.c ou .cc)
      - unité de compilation
        - un module compilé -> un fichier objet (.o)
      - cohérence des fonctionnalités & niveaux de traitement
        - un module regroupe des fonctionnalités de même type
        - les niveau de traitements sont proches
      - remarque : le module fait tout (confus)

Modules & librairies en C/C++

- Programmation modulaire - principes :
  - séparation implémentation / interfaces
    - développeur : doit maîtriser son **implémentation**
    - utilisateur : ne voit que la partie utile du code ie **l'interface**
    - **interface : contrat entre le composant et son utilisateur**
    - interface en C/C++: header (fichier .h)
    - implémentation : .cc, .c
  - compilateur, header et modules :
    - header : ne contient que des déclarations
    - module : implémentation qui inclus les headers nécessaires

Modules & librairies en C/C++

- modules et en-têtes
  - les headers contiennent :
    - des déclaration de macros & constantes
    - de types, de classes
    - des prototypes de fonctions
    - variables et fonctions externes
  - les modules :
    - contient l'implémentation
    - inclus les headers nécessaires
    - les détails d'implémentation sont « privés »

Modules & librairies en C/C++

- Exemple :
- Un programme de manipulation de fichiers sons :

- un module entré-sorties : son_io.cc
- un module interface graphique : son_gui.cc
- un module traitement du signal : son_dsp.cc
- un module principal : son_main.cc
- Un ensemble de déclarations globales son_dec.h
- Les prototypes des modules son_io, son_gui, son_dsp dans les headers son_io.h, son_gui.h, son_dsp.h.
- Les implémentations son_io.cc, son_gui.cc et son_dsp.cc incluent les déclarations générales son_dec.h et leurs .h respectifs.

Modules & librairies en C/C++

- Variables externes

- On peut définir des variables globales à un module. Si un module veut accéder à une variable d'un autre module, il doit déclarer la variable comme une variable externe.
- La définition d'une variable globale n'est pas une déclaration, et doit donc être faite dans un fichier d'implémentation. Par contre, sa déclaration doit être faite dans un fichier d'en-tête. Dans ce cas, l'inclusion de l'en-tête du module rendra la variable visible dans le module.
- L'accès externe à une variable globale d'un module peut être interdit en déclarant la variable en *static*. Dans ce cas, la variable n'est pas « externalisée ».
- Attention : Plusieurs inclusions de variables statiques provoque plusieurs définitions de ces variables qui se masquent mutuellement.

Modules & librairies en C/C++

```
// t.h

extern void aff(void);
static int i;
::::::::::::::

//t.c

#include <stdio.h>

#include "t.h"

void aff(void){

 i=0;

 printf(" i = %d\n",i);

}
```

```
// t_main.c

::::::::::::::

#include <stdio.h>

#include "t.h"

main(){

 i=2;

 aff();

 printf(" i = %d\n",i);

}
```

Modules & librairies en C/C++

- Fonctions externes
  - fonction externe au module, invoquée dans le module
  - par défaut, toutes les fonctions sont externes
  - il faut inclure ces déclarations dans tout module utilisant les des fonctions externes.
  - L'accès externe à une fonction d'un module peut être interdit en déclarant la fonction en *static*.
  - *Exemple : le module son_gui appelle les fonctions de son_io. son_gui.c inclus son_io.h.*

Modules & librairies en C/C++

- éviter les déclarations multiples
- commencer un header par le test d'une variable de compilation définie lorsque le fichier est effectivement inclus.
- Si le header est effectivement inclus, la variable est définie
- **Exemple :**

fichier son_io.h

```
#ifndef  _SON_IO_INCLUDED
#define SON_IO_INCLUDED
.......
#endif
```

Modules & librairies en C/C++

- Compilation séparée des modules
  - *.h + *.c -> .i préprocesseur
  - *.i -> *.s compilation
  - *.s -> *.o assemblage
  - *.o -> exécutable : édition de liens
    - les modules et les librairies sont liées ensembles
    - des segments de code sont ajoutés automatiquement pour permettre l'exécution dans un contexte particulier (console, etc...)

Modules & librairies en C/C++

- Compilation avec gcc :
  - compiler un module seul :
    - g++ -c [opts] <source du module>
- Exemple :
  - gcc -c -o son_gui son_gui.cc
- Regrouper plusieurs modules (Edition de liens):
- gcc [options] -o <cible> <liste d'objets>

Modules & librairies en C/C++

- librairies :
  - principe : collection de modules
  - 2 options :
    - archive (lib<id>.a)
      - simple concaténation de modules compilés
      - à l'édition de liens, les modules utilisés sont extraits et linkés au code client
    - librairie dynamique (dll, .so,...):
      - librairies partagées, code ré-entrant ie exécuté simultanément par plusieurs process

Modules & librairies en C/C++

- Fabriquer une librairie :
  - compiler les modules (.o)
  - archive : commande ar :
    - ar rv lib<id>.a <liste de modules>
  - dynamique :
    - les modules doivent être compilés avec l'option -shared
    - et assemblés avec la commande *ar*
    - la variable LD_LIBRARY_PATH précise le chemin de recherche des librairies dynamiques
    - éditeur de liens dynamiques : ld

Librairies en C/C++

- Exemple :

  gcc -c son_gui.cc

  gcc -c son_io.cc

  ar rv libson.a son_gui.o son_io.o

  gcc son_main.cc libson.a

  ou

  gcc som_main.cc -lson -L .