# TP 1

Principes des Systèmes d'Exploitation

IUT Montpellier

# Objectifs de la séance

- Rappels sur le langage C
- Savoir organiser un projet C
- Savoir compiler un programme C
- Implémenter les ensembles creux d'entiers en C

# Rappels

- La phase de compilation d'un programme C à pour objectif de transformer des fichiers sources en:
  - un binaire exécutable
  - une librairie binaire statique (.a)
  - une libraire binaire dynamique (.so)
- En réalité, cette phase comprend deux étapes majeures:
  - Le pré processeur
  - La compilation du C
- Le pré-processeur
  - Un outil de traitement syntaxique des fichiers qui indépendant du langage C
  - Le préprocesseur traite les commandes comme #include et #define et #ifdef
  - Le résultat est un fichier source *pure* C
- Le compilateur C
  - Réalise le parsing du fichier source C, et donc en texte claire, et produit une série de fichiers intermédiaires comme des fichiers assemleurs (.s) et des fichiers objets (.o)
  - Le résultat final est le binaire binaire:
    - exécutable
    - libraire

# Structure d'un projet C

- Il est conseillé d'avoir une bonne organisation afin de bien gérer son projet en C surtout si celui-ci comporte plusieurs modules et fichiers
- Nous proposons l'organisation suivante:
  - ./include: va contenir tous les fichiers en-tête (.h)
  - ./src: va contenir tous les sources (.c)
  - ./test: va contenir les tests
  - ./obj: destination des fichiers objets (.o)
  - Makefile: Fichier qui va décrire les dépendances et la recette pour construire le fichier final avec la commande *make*

# Exercice 1

- Nous allons réaliser notre premier projet intitulé: *'sayhello'*
- Créer un répertoire 'sayhello' et organiser sa structure interne comme vu précédemment
- Créer un fichier en-tête 'sayhello.h' qui devra déclarer la fonction suivante:
  - `void hello(void);`
- Créer un fichier source 'sayHello.c' et implémenter la fonction *hello* pour afficher un message "Hello World" sur la sortie standard
- Créer un fichier source *'main.c'* qui va définir la fonction *main* et appeler la fonction *hello*
- Réaliser le Makefile pour compiler automatiquement le projet

# Instructions du langage C

- Cet exercice simple va nous permettre de mettre en oeuvre les différentes instructions du langage C

# Algorithmique en C

- Nous souhaitons développer une librairie pour réaliser un algorithme donné par une spécification
- Voici la spécification de l'algorithme:
  - ```
    algo(a, b):
    tant que NOT (a*b=0) faire
      si a>b alors a := a-b
      sinon b := b-a
      fsi
    ftant
    si a=0 alors retourne(b)
    sinon retourne(a)
    fsi
    fin algo
    ```
    ```

# Exercice 2

- A faire
  - Créer un répertoire 'algolib' et organiser sa structure interne
  - Créer un fichier en-tête 'algolib.h' et proposer une signature pour la fonction: *algo*
  - Réaliser une implémentation dans un fichier *algolib.c*
  - Tester votre fonction.
  - Que réalise cet algorithme ?

# Présentation du projet

- Tout au long de l'année, nous allons réaliser un véritable moteur de jeu, avec un rendu ASCII, en utilisant les concepts du cours de systèmes d'exploitation.
- Mais avant cela, nous devons préparer le terrain en réalisant les structures de gestion des données.
- Les ensembles d'entiers sont des structures de base très utiles. Nous pouvons par exemple les utiliser comme des indexes vers des objets.
- Une implémentation efficace de cette structure va nous permettre d'améliorer les performances.

# Introduction

- Nous pouvons distinguer deux cas pour nos ensembles d'entiers:
  - Les ensembles creux
  - Les ensembles denses
- Nous sommes dans le cas creux si:
  - Nous connaissons la valeur maximale de nos entiers
  - Nous pouvons avoir un trou entre deux valeurs
  - Voici un exemple d'un ensemble d'entier creux: $1, 4, 5, 9$
- Nous sommes dans le cas dense si:
  - Nous connaissons la valeur maximale de nos entiers
  - Les valeurs sont très proches les unes des autres
  - Voici un exemple d'un ensemble d'entier creux: $1, 2, 3, 4, 6$
- La connaissance *apriori* du cas d'utilisation va nous permettre de réaliser une implémentation plus efficace en utilisant certaines hypothèses

# Les ensembles creux d'entier

- Votre objectif est de réaliser une implémentation des ensembles creux d'entiers
- Un ensemble creux d'entier est une structure ingénieuse qui va utiliser un 'index' interne pour rendre plus efficace les opérations suivantes:
  - insérer une valeur
  - effacer une valeur
  - rechercher une valeur
  - effacer toutes les valeurs

# Le principe

- Le principe est d'avoir une structure avec les éléments suivants:
  - maxValue: un entier qui représente la valeur max que peut contenir cet ensemble
  - dense[] : un tableau qui va contenir la valeur des éléments de l'ensemble
  - sparse[]:un tableau qui va contenir les indexes à lire dans le tableau dense[]
  - population: un entier qui donne le nombre d'éléments actuellement contenus dans l'ensemble
  - capacity: le nombre max d'éléments que peut contenir cet ensemble

# Les opérations

- insert(x)
- search(x)
- delete(x)
- clear()

# Opération Insérer

- Soit $x$ l'élément à insérer.
- Si $x >= maxValue$ ou $population >= capacity$ alors retourner sans rien faire.
- Insérer $x$ dans dense[*population*]
- Incrémenter *population* de 1
- Mettre la valeur *population* dans sparse[x]

# Opération search(x)

- Si dense[sparse[$x$]] == $x$ alors retourner 1 sinon retourner -1

# Opération delete(x)

- Pour effacer un élément nous le remplaçons avec le dernier élément inséré dans dense[]:
  - Mettre à jour l'index du dernier élément dans sparse[]
  - décrémenter population de 1

# Opération clear()

- il suffit de mettre population à 0

# Exemple 1

- Etat initial
  - Supposons que nous souhaitons gérer un ensemble creux qui peut contenir des entiers de 0 a 10 d'une capacité de 4
  - Cet ensemble au départ vaut: $\{3, 5\}$
  - Voici sa representation
    - maxValue: 10
    - capacity: 4
    - dense = [3,5,*,*]
    - sparce = [*,*,*,0,*,1,*,*,*,*]
    - population: 2
- Insertion
  - Après insertion de la valeur 7
  - Voici sa représentation
    - maxValue: 10
    - capacity: 4
    - dense = [3,5,7,*]
    - sparce = [*,*,*,0,*,1,*,2,*,*]
    - population: 3

# Exemple 2

- Insertion
  - Après insertion de la valeur 4
  - Voici la représentation
    - maxValue: 10
    - capacity: 4
    - dense = [3,5,7,4]
    - sparce = [*,*,*,0,3,1,*,2,*,*]
    - population: 4

# Exemple 3

- retirer
  - Après delete(3)
  - Voici la représentation
    - maxValue: 10
    - capacity: 4
    - dense = [4,5,7,*]
    - sparce = [*,*,*,*,0,1,*,2,*,*]
    - population: 3

# Exemple 4

- clear
  - Après clear()
  - Voici la représentation
    - maxValue: 10
    - capacity: 4
    - dense = [4,5,7,*]
    - sparce = [*,*,*,*,0,1,*,2,*,*]
    - population: 0

# Implémentation des ensembles creux d'entiers

- A faire:
  - Consulter le fichier datamgr.h
  - Compléter le fichier sparseIntegerSet.c
  - Compiler et Lancer le test pour vérifier votre implémentation