Objectifs du projet

- Ecriture de code en langage C
  $\rightarrow$ renforcer la pratique de ce langage
- Opérations de bas niveau sur la mémoire (manipulations de bits), structures, listes et arbres, entrées/sorties sur les fichiers.
- Structure de mini-projet :
  - travail en équipe : quadrinômes
  - temps plein (sur une semaine)
  - autonomie
- Evaluation : soutenance/démonstration du logiciel

Organisation

Planning :

mercredi 25/05 matin : **présentation**	[amphi 001]
mercredi 25/05 après-midi : **package-merge** en 2 groupes	[035+037]
reste du temps : travail dans les salles de td et chez vous	
salles : 013 + 035 + 037 + 039 + 257	
mardi 31 mai après-midi : **soutenances**	
mercredi 1er juin matin : examen C	

Encadrement :

Philippe.Waille@imag.fr, Jeremy.Wambecke@imag.fr

page Web (placard électronique) :

`http://www-ufrima.imag.fr/INTRANET/placard/INFO/ricm3/PROJET_C/`

Thème du projet : compression sans pertes

→ Au menu :

- Compression de Huffman (méthode statique)
- Variante package-merge : arbre de hauteur limitée
-
  - files à priorité
  - (re)construction d'arbre canonique
  - lecture/écriture de fichiers binaires
- Prétraitement données, compactage représentation arbre : RLE + MTF
- Expérimentation sur divers formats de données : textes, programmes (source et binaire), images, etc.
- Statistiques de compression (profondeur de l'arbre, types de documents)

Problèmes spécifiques et outils fournis

- Compréhension des algos /vérification des résultats : vous disposez d'une version exécutable de calcul d'arbre en package-merge.
- Test des algos avant de disposer de routine d'E/S d'entiers de tailles quelconques : simuler dans un premier temps avec chaînes ASCII et/ou bibliothèque d'entrées/sorties binaires.
- Observation du contenu binaire des fichiers compressés : commande unix **od -t x1** ou **hexdump -C**, programme exécutable **bitdump**.

Mesure de compression

Préciser l'indicateur de mesure :

1) rapport de taille : compressé = X

$Taille\_fichier\_compressé / Taille\_fichier\_initial$

2) taux de compression : supression de X

$1 - Taille\_fichier\_compressé / Taille\_fichier\_initial$

Objectifs

1. réversibilité vérifiée : décompression(compression (f)) = f
2. accepte des fichiers de contenu binaire (un bon test : compresser le binaire exécutable du compresseur) : cf avertissement getchar.
3. vraie compression avec gestion des lectures/écritures de bits
4. lmax paramétrable à l'exécution : appliquer
   Huffman pour trouver Lmax_optimal
   puis package-merge si Lmax_optimal > Lmax
5. transmission de l'arbre en table des longueurs compressée
6. choix à l'exécution de prétraitement ou non du contenu par RLE / MTF
7. statistiques de taille et de temps (types de contenus, impact de Lmax)
8. outils de debug/traçage du fonctionnement des algos

Version minimale démontrable

1. réversibilité : obligatoire
2. fichiers de texte uniquement, mais passer du contenu binaire augmente beaucoup l'utilité du compresseur.
3. pas de gestion des bits : simulation par '0' et '1' ASCII ou utilisation de binio (n'empêche pas de calculer les taux de compression en vrai).
4. Huffman uniquement, possibilité de ne pas compresser pour Lmax $>$ 64.
5. format de transmission de l'arbre quelconque non optimisé
6. pas de prétraitement du contenu
7. mettez en valeur ce qui marche avec des exemples et des statistiques
8. pensez mécanismes de trace/debug commutables et réutilisables en démo.

Etapes : chemin critique

A chaque nouvelle version fonctionnelle : tester/archiver (svn,git)

Doit être immédiatement mobilisable en démo en cas de problème dans les versions suivantes.

Pas de course au volume de code : exploitez bien en démo ce qui fonctionne déjà.

Exhaustivité de tests, diversité jeux d'essais, analyse des résultats aussi importants et "payant" que de coder plus de fonctionnalités mal testées/évaluées/présentées.

Se concentrer d'abord sur les prérequis fondamentaux :

- tables longueurs vers arbre/codes canoniques
- arbre de Huffman
- conversion Huffman vers longueurs et arbre canonique

Penser organisation initiale en modules permettant de greffer les fonctionnalités supplémentaires à interface inchangée :

- gestion des bits
- calcul d'arbre
- main enchaînant les étapes
- ...

Gestion du temps

Garder au moins 1/2 journée pour préparer la démonstration.

Figer le code : cf dernière petite modification qui plante tout !

Quand l'échéance approche : privilégier les ajouts les moins risqués :

algos les simples : MTF plutôt que package-merge

sans impact sur le cœur du code : RLE sur contenu plutôt que sur compression de la table des longueurs.

Soutenances/Démonstration

- Déroulement :
  - démonstration : $\sim$ 20 mn
  - questions : $\sim$ 15 mn
- Objectifs : montrer ce qui fonctionne (cf comme à un client) Montrer donc que la base fonctionne et vendez ce que vous aurez fait au delà. Pour celà :
  - préparer des jeux d'essai
  - prévoir le déroulement de la présentation
  - travailler le discours
- Démo des fonctionnalités, pas revue de code C !