Auteur : Pierre Bélisle
Travail en équipe d'au maximum 4 personnes.

# 1. Objectifs

Ce travail a pour principal objectif de mettre en pratique la syntaxe C et les différentes notions enseignées jusqu'à présent : commandes au préprocesseur, utilisation de modules standards, et de modules externes, variables, boucles (*while*), sélections (*if-else*), l'utilisation de fonctions existantes et l'implémentation de fonctions et la manipulation de bits à l'aide d'un module. Il vous est demandé de factoriser (découper en fonctions) votre programme au maximum en créant les différents sous-programmes nécessaires à la réalisation du problème énoncé plus loin.

Vous devez aussi vous initier aux tests unitaires et aux macros-fonctions.

# 2. Description générale du problème

On désire réaliser une application qui enregistre les succès/échecs d'un groupe d'élèves (entre 5 et 32 élèves) pour 3 évaluations. L'utilisateur entre d'abord le nombre d'élèves du groupe. Ensuite, l'application permet : (1) d'inscrire les 3 résultats d'un élève en particulier; (2) de visualiser le nombre de succès de chaque test et le nombre d'élèves dont les résultats ne sont pas entrés; et (3) d'afficher les résultats de tous les élèves.

# 3. Déroulement et détails sur l'exécution demandée

Au départ, le nombre d'élèves est demandé et il doit être entre 5 et 32 ou 0 pour annuler. Ensuite le menu suivant est affiché :

```
Combien d'élèves passent les tests ? : 5

1 : Saisir résultats des tests
2 : Afficher les statistiques
3 : Afficher les résultats
4 : Quitter
```

Après avoir saisi et validé le choix de l'utilisateur, il faut exécuter l'option. Voici le détail de chaque option.

### 3.1 Option QUITTER

Si l’option "quitter" est choisie, le programme confirme que l’utilisateur veut bien quitter. S’il répond ‘o’ ou ‘O’, le programme se termine avec un message de fin. S’il répond ‘n’ ou ‘N’, le programme ne se termine pas. Tout autre caractère est refusé et le choix est redemandé.

### 3.2 Option SAISIE

Vous demandez un numéro d’élève valide par rapport au nombre d’élèves saisi au départ et vous demandez les résultats aux 3 tests qui devront être soit 0 soit 1. Tout autre résultat est invalide et la valeur doit être redemandée.

Si un résultat a déjà été saisi pour un élève, il sera remplacé par la nouvelle saisie.

**Exemple**

```
Entrez le numéro de l’élève : 0

Entrez le résultat du test 1  9   //invalide
Entrez le résultat du test 1  0
Entrez le résultat du test 2  1
Entrez le résultat du test 3  1
```

### 3.3 Option STATS

Si l’option 2 est choisie, vous affichez le nombre d’élèves qui ont eu 0, 1, 2 ou 3 tests réussis. Vous devez distinguer ceux qui n’ont pas encore été entrés de ceux qui ont eu 0 aux 3 tests.

**Exemple**

```
Nombre avec 0 test réussi : 0
Nombre avec 1 test réussi : 0
Nombre avec 2 tests réussis : 1
Nombre avec 3 tests réussis : 0
Nombre n’ayant pas de résultat encore : 4
```

*** Le total doit correspondre avec le nombre total d’élèves saisi au départ.

### 3.4 Option AFFICHE

Si l'option 3 est choisie, vous affichez les numéros d'élèves et leurs résultats. S'il y a plus de 2 des trois tests réussis, une étoile est affichée à côté des résultats. Si le résultat est 000 avec un # c'est que le résultat n'a pas encore été saisi pour cet élève.

**Exemple**

```
Numéro de l'élève          Résultat
-----------------          --------
    0                         101     *
    1                         100
    2                         011     *
    3                         000
    4                         000     #
```

*** On doit comprendre que les élèves 0 et 2 ont réussi 2 des 3 tests, l'élève 1 n'en a, réussi qu'un, l'élève 3 n'en a réussi aucun et on n'a pas entré encore les résultats de l'élève 4.

## 4. Description du problème

Nous avons besoin jusqu'à 96 résultats de succès. Si nous utilisons une variable pour chaque résultat, ça fait beaucoup de variables et beaucoup d'instructions pour traiter la bonne variable selon le choix de l'utilisateur :

```
if(choix == 0)
   //agir sur les variables de l'élève 0
else if (choix ==1 )
  //agir sur les variables de l'élève 1
...
32 fois
```

D'autre part, on remarque que le résultat d'un succès peut être représenté par 1 seul bit, tel que 0:échec, 1:succès. Or, les bits d'un seul entier de 32 bits suffisent pour mémoriser les résultats d'un test pour tout le groupe.

## 5. Solution

Nous allons utiliser les 32 bits d'un entier non signé. Chaque position d'un bit représente l'élève et chaque bit représente s'il a passé ou non le test. Il y aura donc 3 entiers (de 32 bits), un par test. De plus, un quatrième entier sera nécessaire pour retenir si le résultat a été saisi au moins une fois ou non (pour le # de l'affichage).

La position du bit 0 est celle la plus à droite. L'exemple suivant, qui n'utilise que 5 bits pour 5 élèves, montre la mémoire que l'option AFFICHE sur l'exemple décrit en **3.4**.

| | |
|---|---|
| test1 | 00000000000000000000000000000011 |
| test2 | 00000000000000000000000000000100 |
| test3 | 00000000000000000000000000000101 |
| Déjà saisi | 00000000000000000000000000010111 |

Les traitements sur les bits sont réalisés avec un module qui vous est fourni : op_bits. Lisez bien comment utiliser chaque fonction dans les commentaires du fichier op_bits.h. Remarquez aussi que les fonctions de op_bits ne modifient pas l'entier passé en paramètre (comme il s'agit d'un passage par copie), mais plutôt retourne un nouvel entier correspondant à l'entier modifié par l'opération. Ainsi, il ne faut pas oublier, par exemple, d'actualiser la variable avec le retour de la fonction. Cette stratégie implique un code du type :

variable = operation(variable,...);

## 6. Développement

Nous suggérons une approche de développement descendante à partir du programme principal. Commencez par écrire la boucle principale du main() qui demande une confirmation de l'utilisateur lorsqu'il quitte l'application avec l'option 4 du menu.

Début

Mettre un booléen à faux

Tant que le booléen est à faux

Appeler la procédure principale de l'application

Affecter le booléen selon la réponse de l'utilisateur s'il veut vraiment quitter.

Fin de la boucle

Afficher un message de fin

Fin

---

## 6.1 Procédure principale de l’application

Dans ce sous-programme, vous déclarez vos 4 entiers et tout ce dont vous aurez besoin en cours de route (choix de l’utilisateur, nombre d’élèves …). Utilisez des sous-programmes pour des sous-tâches spécifiques d'une certaine complexité (on ne réalise pas de sous-programme, par exemple, qui contient qu'une seule instruction, comme il est plus clair/efficace d'appeler cette instruction directement).

### Pseudocode

**Début**

Saisir le nombre d’élèves entre 5 et 32 (ou 0 pour annuler).

Si le nombre est différent de 0

Tant que l’utilisateur n’a pas choisi l’option quitter du menu

Offrir le menu principal

Saisir et valider choix

Si le choix est différent de l’option "quitter"

Si le choix est d’afficher les stats, appeler le sous-programme qui affiche les stats

Sinon si le choix est d’afficher les résultats, appeler le sous-programme qui affiche les résultats.

Sinon

Saisir le numéro de l’élève.

Obtenir les résultats aux tests (**voir 6.2**).

Ajuster les résultats de bon élève à partir de la valeur reçue de la fonction précédente (C’est ce qui **ne** pourra **pas** être fait dans un sous-programme).

Fin

Fin de la boucle

Fin (si le nombre est différent de 0)

**Fin**

## 6.2 Procédure pour obtenir les résultats aux tests

Cette procédure demande à l'utilisateur le résultat de chacun des 3 tests (0:échec, 1:succès), et retourne une valeur (unsigned char) variant de 0 à 7 qui indique les résultats obtenus selon le tableau suivant, où on observe que Valeur=r1*4+r2*2+r3 avec r1 le résultat du test 1; r2 le résultat du test 2 et r3 le résultat du test 3.

Remarquez que le paramètre de la méthode get_bits de op_bits peut correspondre à la valeur de retour de cette sous-procédure, bien qu'elle soit de type *unsigned char*, étant transtypée en *unsigned int* sans perte d'information.

| Valeur | Test1 | Test2 | Test3 |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 0 | 0 | 1 |
| **2** | 0 | 1 | 0 |
| **3** | 0 | 1 | 1 |
| **4** | 1 | 0 | 0 |
| **5** | 1 | 0 | 1 |
| **6** | 1 | 1 | 0 |
| **7** | 1 | 1 | 1 |

## 6.4 Test unitaire et débogage

Réalisez des procédures de test-unitaires validant les procédures de votre application. Ces procédures peuvent être réalisées à même le fichier principal du projet (contenant le main), ou dans un module séparé. Vous pouvez prendre la stratégie de votre choix.

Les procédures à tester sont celles qui n’interagissent pas avec l’utilisateur et dont le résultat est prévisible avec des entrées précises. Les procédures de tests font partie du  travail à rendre.

## 6.5 Étape d'optimisation: utilisation de MACRO-fonctions

*Cette étape sera réalisée à la dernière semaine allouée au laboratoire*. On assume ici un scénario fictif : une fois votre programme fonctionnel, vous le

tester et désirer augmenter la rapidité d'exécution des traitements pour obtenir des bits et modifier des bits (actuellement réalisé avec les méthodes get_bit et set_bit). Pour ce faire, vous décidez d'utiliser des MACRO-fonctions. Vous devez donc, dans le fichier « ***op_bits.h*** », déclarer deux MACRO-fonctions (GET_BIT(n, o), SET_BIT(n, o) [1] qui remplaceront les fonctions du même nom de ce module. Ensuite, dans votre code, vous remplacez TOUS les appels de ces fonctions par des appels aux versions "MACRO", où la constante MACRO permettra d'activer l'utilisation des macro-fonctions (1: macro-fonctions activées; 0: anciennes fonctions utilisées).

```
#if(MACRO)
        … = SET_BIT(…);
#else
        … = set_bit(…);
#endif).
```

*Remarque :* dans le cadre de ce travail, la différence en temps n'apparaîtra pas significativement que vous utilisiez les fonctions ou les MACRO. Vous pourrez comparer le temps d'exécution des deux approches en utilisant le module « ***t_chrono*** » fourni avec cet énoncé. La comparaison peut être réalisée dans une procédure de test qui obtient et modifie les bits d'un entier non signé. Vous démarrez le chrono, exécutez un million de fois avec les fonctions et arrêtez le chrono. Même chose ensuite avec les MACRO. Comparer les temps. Y a-t-il un gain ?

## 7. Réalisation et commentaires

- Mettre le nom des contributeurs pour chaque procédure dans le code.
- Il est interdit de dédier un-e étudiant-e à la **seule** écriture ou la **seule** révision des commentaires : l'étudiant qui réalise un code est celui qui commente ce même code. Les commentaires doivent ainsi être écrits au fur et à mesure que le code est écrit.
- Les tests aussi doivent être commentés.
- Vous aurez à saisir des entiers et des caractères alors n'oubliez pas fflush(stdin).

[1] Les paramètres nombre et ordre ont été abrégés en n et o pour raccourcir la ligne lorsque vous l'écrirez.

*Il n'est pas permis de copier-coller ni de consulter le code ne provenant pas d'un membre de votre groupe. En cas de problème, n'hésitez pas à consulter le chargé de laboratoire aux séances ou par courriel. Cette infraction de nature académique détectée sera dénoncée au comité prévu pour ces cas.*

## 8. Barème de correction

**Exécution et tests** 40%

| **Élément** | **Pondération** |
|---|---|
| Fonctionnement du menu général selon l'entrée choisie | 5 |
| Fonctionnement de "Saisir résultats des tests" | 6 |
| Fonctionnement de "Afficher les statistiques" | 7 |
| Fonctionnement de "Afficher les résultats" | 7 |
| Mise à jour correcte des statistiques et des résultats | 9 |
| Les procédures de tests valident bien le fonctionnement des procédures testées | 4 |
| Robustesse aux mauvaises entrées de l'utilisateur | 2 |

**Qualité de programmation** 60%

| **Élément** | **Pondération** |
|---|---|
| Organisation des sous-programmes **prototypes-main-sous-programmes / modules.** Le tout judicieusement distribué. | 10 |
| Organisation du code en sous-procédures pertinentes, il est facile de comprendre le mandat de chaque bloc de code, et comment il le réalise. Lié à l'efficacité du code | 20 |
| Lisibilité du code : les noms des éléments sont bien choisis selon les normes, l'indentation est claire, les retours de ligne et l'espacement bien utilisés | 10 |
| Qualité de la documentation : commentaires complets, faciles à comprendre, sans fautes d'orthographe. Les références sont clairement indiquées et bien citées. | 20 |

*Bon travail!*