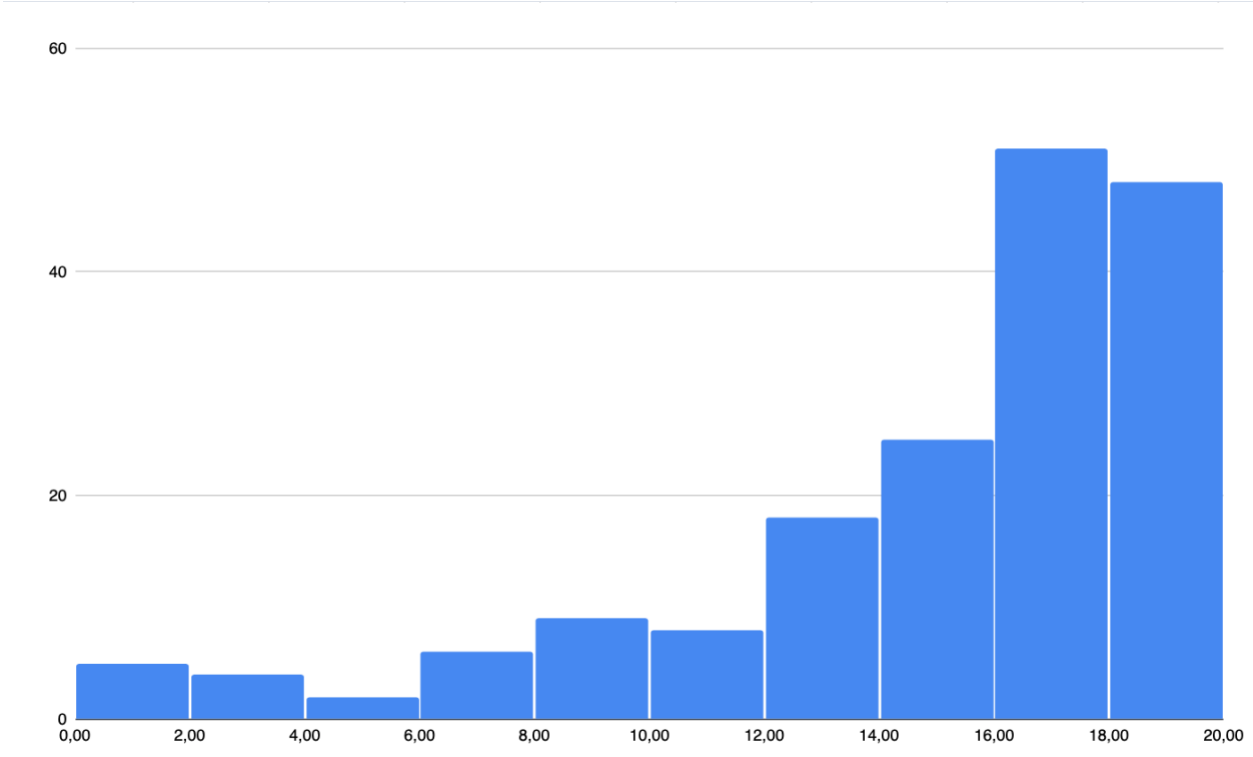
INFO0952

Feedback Devoir 1
2019-2020

Dernière mise à jour le 4 novembre 2019

Analyse des résultats

- 199 devoirs soumis
- 173 réussites (cote $>$ 10), 23 maximums (20)
- Moyenne : 15,4, médiane : 16,9
- 14 codes ne compilent pas
- 72 provoquent des warnings à la compilation (c'est beaucoup trop !)



Résultats par fonction

Sur 199 soumissions :

- `search_pattern` : 64 (!) solutions correctes
- `rotate` : 125 solutions correctes
- `decrypt` : 121 solutions correctes
- `test_all_key_permutations` : 113 solutions correctes

Remarques générales

- Plagiats : Nous avons répéré des similitudes entre plusieurs paires de devoirs. Certains étudiants ont été sanctionnés, d'autres seulement avertis. Nous serons moins tolérants pour la suite. N'échangez jamais de code entre vous !
- La cote est essentiellement basée sur l'exactitude de vos fonctions, testée via des scripts automatiques. Certains codes ne compilaient pas, même après une tentative (rapide) de correction manuelle, et ont donc malheureusement du être sanctionnés d'un zéro. Il est crucial de fournir un code qu'on peut compiler et tester. Il est impossible de corriger manuellement 199 devoirs/projets.

Remarques générales

- Trop de warnings : affichez les et corrigez les ! Ils annoncent souvent des erreurs. Quelques exemples rencontrées : “=” au lieu de “==”, variables non initialisées, prototypes manquants (!), arguments de fonctions ou variables non utilisés, pas d'instruction `return` pour une fonction non `void`, etc.
- Beaucoup d'erreurs dues au non traitement du caractère `\0` à la fin des chaînes de caractères (tableau alloué trop petit, oubli d'ajout de cette valeur avant d'appeler `strlen` ou `strcpy`, etc.).
- En C99, on ne peut pas définir une fonction à l'intérieur d'une fonction, même si c'est accepté par certaines versions de `gcc`. Ne le faites pas dans ce cours !

Une solution correcte pour search_pattern

```
int search_pattern(char *pattern, char *message) {

  int lpattern = strlen(pattern);
  int lmessage = strlen(message);

  for (int i = 0; i< (lmessage-lpattern+1); i++) {
    int j = 0;
    while ((j < lpattern) && (message[i+j] == pattern[j]))
      j++;
    if (j == lpattern)
      return i;
  }
  return -1;
}
```

- Invariant boucle externe : le pattern n'apparaît pas aux positions $0 : \texttt{i} - 1$. Le gardien vérifie l'apparition du pattern à la position i.
- Invariant boucle interne : $\texttt{pattern}[0 : \texttt{j} - 1] = \texttt{message}[\texttt{i} : \texttt{i} + \texttt{j} - 1]$

Erreurs fréquentes

- Mauvaise borne supérieure dans la boucle externe :
  - `lmessage-lpattern` : pattern pas détecté en fin de message
  - `lmessage` : on sort des bornes du tableau !
- Traitement particulier du premier caractère :

```
if (pattern[0] == message[i]) {
   int j = 1;
   while (...) {
      ...
   }
}
```

  Rend le code plus complexe et source d'erreur dans le cas d'un pattern de longueur 1.
- Utilisation de la fonction `strstr` de `string.h` : quand on vous demande d'implémenter une fonction, vous ne pouvez pas utiliser une fonction toute faite faisant la même chose. En cas de doute, demandez.

Erreurs fréquentes

Seulement 115 codes donnent la bonne réponse pour l'appel : `search_pattern("aabc","xxxaaabcxxx")`

Un exemple de solution erronée :

```
int search_pattern(char *pattern, char *message) {
  int lpattern = strlen(pattern);
  int lmessage = strlen(message);
  int i = 0;
  int j = 0;
  while (i<lmessage && j<lpattern) {
    if (message[i] == pattern[j]) {
      i++;
      j++;
    } else {
      i++;
      j=0;
    }
  }
  if (j == patlength)
     return i-j;
  else
     return -1;
}
```

Une solution pour `rotate`

```c
void rotate(char *tab, int shift, int n) {
    int d = 0;
    if (shift < 0) {
        d = n - (-shift % n);
    } else {
        d = shift % n;
    }
    char *new_tab = malloc(n * sizeof(char));
    for (int i = 0; i < n; i++) {
        new_tab[(i + d) % n] = tab[i];
    }

    for (int i = 0; i <n; i++)
        tab[i] = new_tab[i];
    free(new_tab);
}
```

Globalement mieux faite que `search_pattern`. Ne pas oubliez de libérer la mémoire temporaire allouée !

Une solution pour `test_all_key_permutations`

```
char *test_all_key_permutations(char *message, char *pattern, char *key){
    if (test_all_perms_rec(message, pattern, key, 0, strlen(key)))
      return key;
    else
      return NULL;
}

static int test_all_perms_rec(char *message, char *pattern, char *key, unsigned pos, unsigned kl) {
  if (pos == kl-1) { // cas de base
    char *decrypted_message = decrypt(encmessage, key);
    if (search_pattern(pattern, decrypted_message)!=-1) {
      free(decrypted_message);
      return 1;
    } else {
      free(decrypted_message);
      return 0;
    }
  }
  // cas inductif
  for (unsigned i = pos; i < kl; i++) {
    swap((key+pos),(key+i));
    if (test_all_perms_rec(message, pattern, key, pos+1, kl)) {
      return 1;
    }
    swap((key+pos), (key+i));
  }
  return 0;
}
```

Une solution pour `test_all_key_permutations`

Fonction plus compliquée. N'hésitez pas à afficher les patterns testés pour vous convaincre que la fonction est correcte.

Erreurs fréquentes : mauvais placement des `return` (par exemple, oubli du dernier `return 0`).