Aix-Marseille Université
Faculté des Sciences de Luminy

Licence Info – Semestre 4
Lundi 11 Mai 2020

Programmation C et Système — Examen

Rendre le fichier-texte de réponse sur Ametice avant 18h

Le sujet de l'examen prend pour prétexte un programme listant les entrées du répertoire courant, tel que le ferait `ls -a`*, mais triés en ordre inverse, et en excluant les noms contenant des espaces. (L'option* `-a` *inclut tous les fichiers y compris les fichiers cachés commençant par un point).*

Question 1. La fonction `String_dup()` est analogue à `strdup()` et duplique une chaîne `string` en allouant dynamiquement sa copie.

1. Calculer la taille `size` en byte de la chaîne, zéro terminal inclus.
2. Allouer dynamiquement la mémoire nécessaire pour la copie `copy`.
3. Vérifier par une assertion l'allocation de la mémoire.
4. Utiliser une fonction de librairie pour recopier `string` dans la copie allouée.

```
char * String_dup (char const string[]) {
  size_t size= ..................................................
  char * copy= ..................................................
  ..............................................................
  ..............................................................
  return copy;
}
```

Question 2. La fonction `String_charIndex()` retourne l'index de la première occurrence du caractère `character` trouvée dans la chaîne `string`, ou `-1` en l'absence d'occurrence. On décide que le caractère `'\0'` n'est pas un argument valide.

1. Utiliser une assertion pour vérifier que `character` n'est pas le caractère `'\0'`.
2. Utiliser une fonction de librairie pour trouver l'adresse `occ` de l'occurrence.
3. Traiter le cas de l'absence d'occurrence par une clause de garde.
4. Utiliser l'arithmétique de pointeurs pour retourner l'index de l'occurrence `occ` trouvée.

```
int String_charIndex (char const string[], char character) {
  ..............................................................
  char * occ= ...................................................
  ..............................................................
  return ........................................................
}
```

Question 3. Le prédicat `String_containsChar()` teste si la chaîne `string` contient le caractère `character`. On décide que le caractère `'\0'` est un argument valide et que le prédicat renvoie `false` dans ce cas. En réutilisant `String_charIndex()`, compléter son corps sans utiliser de `if`.

```
bool String_containsChar (char const string[], char character) {
  return ........................................................
         ........................................................
}
```

Question 4. Les trois fonctions précédentes vont dans le module `String`, Compléter le fichier d'entête `String.h`, avec sa garde contre l'inclusion multiple. Préciser l'entête à inclure pour que le fichier d'entête `String.h` soit valide.

```
.............................
.............................
#include .....................
char *  String_dup (char const string[]);
int     String_charIndex    (char const string[], char character);
bool    String_containsChar (char const string[], char character);
...............
```

Question 5. Écrire un test `StringTest_dup()` pour la fonction `String_dup()` contenant une assertion. Penser à éviter les fuites de mémoire. Écrire un test `StringTest_containsChar()` pour le prédicat `String_containsChar()` contenant trois assertions couvrant les trois cas principaux.

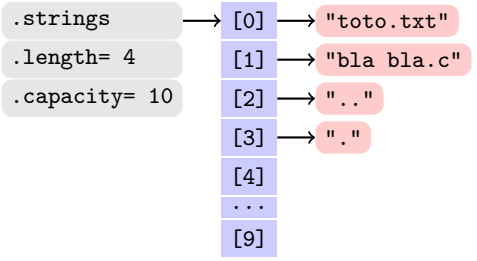
```
void StringTest_dup (void) {
  ..................................................................
  ..................................................................
  ..................................................................
}
```

```
void StringTest_containsChar (void) {
  ..................................................................
  ..................................................................
  ..................................................................
}
```

Question 6. Le type `Content` pointe via son champ `strings` vers un tableau de `length` chaînes. Les chaînes sont allouées dynamiquement, de même que le tableau `strings`. Le tableau a une capacité `capacity` initiale de `MIN_CAPACITY` chaînes. La capacité double quand `length` l'atteint lors d'un ajout de chaîne. Un exemple d'une variable de type `Content` est dessiné ci-dessous à droite. Compléter la définition du type `Content` avec ses champs `strings`, `length`, et `capacity`. Puis définir une macro `MIN_CAPACITY` valant `10`.

```
..................................... {
  ..................................
  ..................................
  ..................................
} Content;

.................................
```



Question 7. La fonction `Content_initEmpty()` initialise `c` en allouant `MIN_CAPACITY` cases pour son tableau `strings`, qui ne contient initialement aucune chaîne. Vérifier par une assertion le succès de l'allocation.

```
void Content_initEmpty (Content * c) {
  ..................................................................
  ..................................................................
  ..................................................................
  ..................................................................
}
```

Question 8. La fonction `Content_doubleCapacity()` double la capacité de `c`. Après avoir calculé sa nouvelle taille en byte `newSize`, réallouer dans `newStrings` le tableau `strings` de `c`. Vérifier par une assertion le succès de la réallocation, puis mettre à jour les champs de `c`.

```
void Content_doubleCapacity (Content * c) {
  ......... newSize= ..........................................
  ......... newStrings= .......................................
  ............................................................
  ............................................................
  ............................................................
}
```

Question 9. Compléter le prédicat `Content_isFull()` qui teste si la longueur du tableau de `c` a atteint sa capacité. Compléter la fonction `Content_addString()` qui rajoute une copie de la chaîne `string` dans le tableau `strings` de `c`. Le tableau double sa capacité lorsque celle-ci est atteinte. La copie est allouée dynamiquement.

```
bool Content_isFull (Content const * c) {
  return ....................................................
}
```

```
void Content_addString (Content * c, char const string[]) {
  if (.....................................) {
    ....................................
  }
  ....................................
  ....................................
}
```

Question 10. La fonction `Content_clean()` libère toutes les ressources allouées dynamiquement par `Content_initEmpty()` et `Content_addString()`. Compléter son corps.

```
void Content_clean (Content * c) {
  for (size_t k= ............ ;  ............ ; ............) {
    ....................................
  }
  ....................................
}
```

Question 11. La fonction `Content_addDirEntryNames()` rajoute dans `c` le noms de toutes les entrées du répertoire ouvert `dir`. Compléter le corps de sa boucle inconditionnelle. La fonction ne filtre pas les entrées `"."` et `".."`, ne trie pas les entrées, et ne ferme pas `dir`. On rappelle par ailleurs que la structure `dirent` possède un champ `d_name`.

```
void Content_addDirEntryNames (Content * c, DIR * dir) {
  for (;;) {
    ............................................................
    ............................................................
    ............................................................
  }
}
```

Question 12. Le prédicat `Content_isRevSorted()` teste si les chaînes de `c` sont triées en ordre inverse. Compléter son corps.

```
bool Content_isRevSorted (Content const * c) {
  for (size_t k= ..... ; ..................... ; ..... ) {
    if ( ..................................................... )
       .....................................................
  }
  .....................................................
}
```

Question 13. On suppose que l'on dispose de la fonction `revCompareAsString()` utilisable conjointement avec `qsort()` pour comparer deux chaînes en ordre inverse. Compléter le corps suivant de la fonction `Content_revSort()` qui trie les chaînes de `c` en ordre inverse. La variable `cellSize` contient la taille en bytes d'une case du tableau à trier. Vérifier à la fin par une assertion que les chaînes sont bien triées en ordre inverse.

```
void Content_revSort (Content * c) {
  size_t cellSize= ...................
  ..............................................................
  ..............................................................
}
```

Question 14. Compléter la fonction `Data_revCompareAsString()` mentionnée à la question précédente. Parmi les choix possibles, où doit-on placer les qualificateurs `const` manquants ?

```
int revCompareAsString (void const * p1, void const * p2) {
  char ..... * ..... * ..... string1= p1;
  char ..... * ..... * ..... string2= p2;
  return ...................................................
}
```

Question 15. On définit `StringPredicate` comme le type des fonctions qui sont des prédicats sur une chaîne. Définir ce type, puis écrire un prédicat `String_isSpaceFree()` de ce type. Ce prédicat teste qu'une chaîne est sans caractère espace et est un simple wrapper sur `String_containsChar()`.

```
.................. StringPredicate ..............................
```

```
..... String_isSpaceFree ( ............................. ) {
  ..............................................................
}
```

Question 16. La fonction `Content_print()` affiche dans le flux `file` un sous-ensemble des chaînes de `c`, à raison d'une par ligne. Lorsque `accept` est non-`NULL`, seules les chaînes acceptées par `accept` sont affichées. Lorsque `accept` est `NULL`, toutes les chaînes sont affichées, comme si `accept` retournait toujours `true`. Compléter la boucle de la fonction en utilisant une clause de garde pour ignorer le tour de boucle lorsqu'une chaîne ne doit pas être affichée.

```
void Content_print (Content const * c,
                    StringPredicate * accept, FILE * file)
{
  for (size_t k= ..... ; ..................... ; ..... ) {
    ..............................................................
    ..............................................................
  }
}
```

Question 17. Compléter ce `main()` qui utilise `Content` pour lister les fichiers du répertoire courant, triés en ordre inverse, en excluant les noms contenant des espaces.

```
int main (void) { // version 1
  DIR * dir= opendir (".");
  if (dir == NULL) { perror ("opendir"); exit (1); }
  Content c;
  ..........................................................
  ..........................................................
  closedir (dir);
  ..........................................................
  ..........................................................
  ..........................................................
  return 0;
}
```

Question 18. On souhaite écrire une deuxième version du programme n'utilisant pas `DIR*` et `Content`, mais utilisant `popen()` qui lance un pipeline de commandes. Compléter l'affectation de `PIPELINE` avec la chaîne contenant le pipeline que l'on aurait écrit sous le shell pour lister les fichiers triés en ordre inverse en excluant les noms contenant des espaces. Compléter la boucle de lecture en utilisant la fonction de lecture de ligne `fgets()`. On supposera qu'un seul `fgets()` suffit pour lire une ligne complète.

```
char const PIPELINE[]= ..........................................
```

```
int main (void) { // version 2
  FILE * stream= popen (PIPELINE, "r");
  if (stream == NULL) { perror ("popen"); exit (1); }
  for (;;) {
    char filename [256];
    ..................................................................
    ..................................................................
    ..................................................................
  }
  pclose (stream);
  return 0;
}
```

Question 19. On souhaite écrire une troisième version du programme qui crée un processus enfant, et se recouvre avec l'image de `sh -c` lançant le pipeline `PIPELINE` via la fonction `execlp()`. En cas d'échec du recouvrement, la raison de l'erreur s'affiche et l'enfant se termine avec un status d'échec. Compléter la première moitié de la version 3 qui concerne l'enfant :

```
int main (void) { // version 3
  pid_t child= fork ();
  if (child == -1) { perror ("fork"); exit (1); }
  if ( ................. ) {
    execlp ( ..................................................... );
    ..............................................................
    ..............................................................
  }
  // version 3 continues below
```

Question 20. Compléter la seconde moitié de la version 3 qui concerne le parent. Faire en sorte qu'il attende la terminaison de son fils unique. Vérifier par une assertion que le processus attendu est bien le fils. Utiliser les macros `WIFEXITED()` et `WEXITSTATUS()` pour initialiser la variable `success()`. Cette variable Booléenne indique que le fils s'est terminé normalement et avec succès. Terminer ensuite le programme avec un status d'erreur basé sur l'état de `success` en utilisant l'opérateur ternaire.

```
  // version 3 continues here
  int info;
  ............... = wait ( ............... );
  ..............................................................
  bool success= ................................................
  return .......................................................
}
```