Universtité Nice Sophia Antipolis
Polytech'Nice Sophia - SI 3

2020 – 2021

NOM: NIGET
PRÉNOM: Tom

Programmation Procédurale

15 janvier 2021

Documents non autorisés. **Durée: 1h30**

Vous pouvez omettre les directives `#include` *dans vos réponses. Vous apporterez un très grand soin à* ***la présentation*** *car elle* ***interviendra dans la notation.*** *Par ailleurs, les solutions trop compliquées seront* ***pénalisées.***

2/2

Question 1: *Suite de Syracuse*

On définit la **suite de Syracuse** pour l'entier $u_0 > 0$ comme suit: le terme suivant de la suite (u_1) est égal à $\frac{u_0}{2}$ si u_0 est pair et à $3 \times u_0 + 1$ dans le cas contraire. Dans le cas général, on a donc:

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3 \times u_n + 1 & \text{sinon.} \end{cases}$$

On admettra ici que les toutes les suites de Syracuse convergent vers la valeur 1. Voyons quelques exemples:

```
Syr(5) = 5 16 8 4 2 1                                  (longueur de Syr(5) = 6)
Syr(8) = 8 4 2 1                                       (longueur de Syr(8) = 4)
Syr(11) = 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1      (longueur de Syr(11) = 15)
```

Écrire la fonction `unsigned syracuse_length(unsigned u0)` qui renvoie la longueur de la suite de Syracuse partant de l'entier strictement positif `u0` (par exemple, cette fonction renverra 15 si elle est appelée avec la valeur 11).

```
unsigned syracuse_length(unsigned u0)
{
  int i;
  for(i=1; u0!=1; i++)
  {
    u0 = u0%2?(3*u0+1):(u0/2);
  }
  return i;
}
```

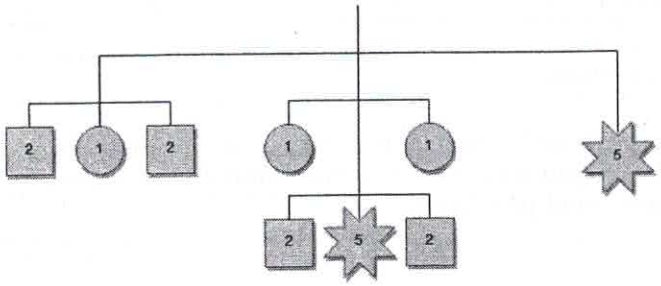
2

Question 2: *Suspension Mobile*

Une **suspension mobile** est définie comme étant soit une décoration, soit un triplet {*left*, *center*, *right*} où *left*, *center* et *right* sont des suspensions mobiles. Le poids des différentes décorations que l'on peut mettre sur un mobile est donné ci-dessous:

```
#define circle 1        // poids d'un cercle
#define square 2        // poids d'un carré
#define star   5        // poids d'une étoile
```

Un exemple de suspension mobile (les poids sont inscrits sur les décorations):



On utilise la structure suivante pour représenter les décorations *et* les suspensions:

```
struct element {
  int decoration;
  struct element  *left, *center, *right;
};
```

Si le champ **decoration** est nul, les champs *left*, *center*, *right* désignent les objets (décorations ou suspensions) accrochées à cet élément de suspension. Si le champ **decoration** a une valeur différente de 0, celle-ci correspond au poids de la décoration (et les champs *left*, *center*, *right* ne servent pas).

Définir à l'aide d'un `typedef` C le type `Element` comme un **pointeur** sur un `struct element`:

1/1

```
typedef struct element* Element;
```

1

Les fonctions `create_suspension` et `create_decoration` permettent de créer une nouvelle suspension ou une nouvelle décoration. Par exemple, le *sous-mobile* de gauche précédent peut être créé ainsi:

```
Element left = create_suspension(create_decoration(square),
                                 create_decoration(circle),
                                 create_decoration(square));
```

Écrire la fonction `Element create_decoration(int w);` permettant de créer une nouvelle décoration de poids `w`:

1/2

```
Element create_decoration(int w)
{
  Element res = calloc(sizeof(*res), 1);
  res->decoration = w;
  return res;
}
```

1

et si NULL ?

Écrire la fonction `Element create_suspension(Element l, Element c, Element r);` permettant de créer une nouvelle suspension avec les trois sous-mobiles `l`, `c` et `r`.

1/2

```
Element create_suspension(Element l, Element c, Element r)
{
  Element res = calloc(sizeof(*res), 1);
  res->left = l;
  res->center = c;
  res->right = r;
  return res; // decoration est déjà à 0 du fait de calloc
}
```

1

Écrire la fonction `int weight(Element mobile)` qui renvoie le poids total d'un mobile. Par exemple, le mobile de la figure précédente a un poids de 21.

1.75/2

```
int weight(Element mobile)
{
  if(mobile->decoration != 0)
    return mobile->decoration;
  return weight(mobile->left) + weight(mobile->center) + weight(
      mobile->right);
}

// Comme ce n'est pas précisé, je suppose que left, center, right != NULL.
// Pour gérer ce cas, rajouter  if(!mobile) return 0;
// au début
```

si NULL ?

1.75

↑ dès tous les cas.

On veut écrire la fonction `int is_balanced(Element m)` qui indique si le mobile est équilibré (comme celui de la figure précédente par exemple).

Définir, **en français**[1], les conditions nécessaires pour qu'un mobile soit équilibré:

0.5/1

Un mobile est balancé si :
- c'est une décoration
- ou bien si le poids à gauche est égal au poids à droite

0.5

pas rendement

[1] ou en anglais, mais pas en C ici.

Écrire, en la fonction `is_balanced`:

```
int is_balanced(Element mobile)
{
  return mobile->decoration || (weight(mobile->left)==weight(mobile->left));
}
```

+ equilibres poutre2

1

1/2

Question 3: *Fonction atexit*

La fonction standard `atexit` est définie comme:

```
int atexit(void (*fct)(void));
```

Cette fonction enregistre le fait que la fonction `fct` doit être appelée lorsque le programme se termine (par exemple en appelant la fonction système `exit`). Les fonctions enregistrées par `atexit` sont appelées dans l'ordre inverse de leur enregistrement. La fonction `atexit` renvoie `0` si l'enregistrement a réussi et `-1` dans le cas contraire. On suppose ici que l'on peut enregistrer au maximum `MAX_FUNC` fonctions par `atexit`.

- Écrire la fonction `atexit`.

```
void (*funcs[MAX_FUNC])(void);
unsigned func_count = 0;
int atexit(void (*fct)(void))
{
  if(func_count == MAX_FUNC) return -1;
  funcs[func_count++] = fct;
  return 0;
}
```

2/2

2

- Écrire le morceau de code du système chargé de l'exécution des fonctions enregistrées par `atexit`.

```
unsigned n = func_count; // les accès sur la pile sont
while (n)                // généralement plus rapide que
  funcs[--n] ();         // sur le tas donc je crée une copie
```

1/1

1

NOM: NIGET PRÉNOM: Tom

Question 4: *Création dynamique de chaînes de caractères*

Écrire la fonction `char *strcreate(int n, char *init);` qui renvoie une nouvelle chaîne de caractères de longueur `n` initialisée avec les caractères de la chaîne `init`. Si `n` est supérieur à la longueur de `init`, les caractères de `init` sont répétés.

```
strcreate(3, "abcd")    ⟶ "abc"
strcreate(9, "abcd")    ⟶ "abcdabcda"
strcreate(20, "+-")     ⟶ "+-+-+-+-+-+-+-+-+-+-"
strcreate(4, ".")       ⟶ "...."
```

Écrire la fonction `strcreate` (on suppose ici que la chaîne `init` n'est pas égale à `NULL` et que les allocations réussissent toujours).

1.5/2

```
char* strcreate(int n, char* init)
{
  char* res = malloc(n);
  char* orig = init;
  for(int i=0; i<n; i++, init++)
  {
    if(!*init) // si fin de init, on reboucle
      init = orig;
    res[i] = *init;
  }
  res[n] = '\0';
  return res;
}
```

Question 5: *Chaînes de caractères sûres*

Le langage C ne vérifie pas que les accès aux chaînes de caractères sont corrects. C'est-à-dire que C ne fait aucun test de bornes lors d'un accès en lecture ou en écriture. À des fins de mise au point, on veut pouvoir faire ces vérifications. Pour cela, on utilise dans nos programmes `string_ref` et `string_set` pour accéder aux chaînes. On suppose ici que `string_ref(s, i)` permet de lire le i^{eme} caractère de la chaîne `s` et que `string_set(s, i, c)` permet d'y ranger le caractère `c` (ces deux fonctions sont définies dans le fichier `safe-string.c`):

```
#include <stdio.h>
#include <stdlib.h>
#include "safe-strings.h"

int main() {
   char ch1[] = "Hello, world!";
   char ch2[] = "abc";

   string_set(ch2, 8, string_ref(ch1, 5));  // <=> ch2[8]=ch1[5] => Erreur
   printf("0: %c\n", string_ref(ch2, 0));   // acces a ch2[0]   => OK
   printf("60: %c\n", string_ref(ch1, 60)); // acces a ch1[60]  => Erreur
   return 0;
}
```

2.25/4

(je n'ai pas eu le détail des points par question, seulement pour la page)

1. Donner une implantation possible de `string_ref` et `string_set` avec des fonctions pour que les accès aux chaînes soient sûrs (i.e. votre implémentation se contentera d'afficher un message d'erreur lorsque l'accès est illégal). *Note: Pour cette question, on n'aura pas de préoccupations d'efficacité.*

Implémentation de la fonction `string_ref`*:*

```
char string_ref( char* s, unsigned i) //renvoie 0 pour s[len(s)]
{
  while(i--)
    if(!*s++){puts("Acces illegal"); return 0;}

  return *s;
}
```

Implémentation de la fonction `string_set`*:*

```
void string_set(char* s, unsigned i, char c)
{ i++;
  while(i--)
    if(!*s++){ puts("Acces illegal"); return ;}

  s[-1] = c; //pour empêcher d'écrire sur le terminateur
}
```

2. Parce que l'utilisation de `string_ref` et `string_set` ralentit fortement l'exécution des programmes, on veut pouvoir désactiver éventuellement les surveillances de mise au point. Donner le corps complet du fichier `safe-string.h` pour que les tests de bornes ne soient effectués par `string_ref` et `string_set` que si la macro `DEBUG` est définie (dans le cas contraire `string_ref` et `string_set` doivent être aussi efficaces que des accès directs aux chaînes de caractères).

```
#ifndef SSTRING_H
#define SSTRING_H

#ifdef DEBUG
#define string_ref(s,i)   (s)[i]
#define string_set(s,i,c) do{(s)[i]=(c);}while(0)
#endif

//en compilation non DEBUG, soit ne pas compiler safestrings.c,
//soit englober son code dans un #ifdef

#endif
```