TD2

September 13, 2022

1 TD1

2 TD2

2.1 Exécutions ligne à ligne de programmes récursifs

On commence par corriger les deux derniers exercices du 1.1 et les 3 exercices du 1.2.

Ecrire la fonction syracuse (cf dernier TD) en récursif.

2.2 Exécutions ligne à ligne de programmes avec pointeurs et tableaux

[2]:
```
void minMax (int longueur, int tab[longueur], int* min, int* max)
{
    assert(longueur > 0) ;
    *min = tab[0] ;
    *max = tab[0] ;
    for (int i = 1; i < longueur; i++)
    {
        if (tab[i] < *min)
            *min = tab[i] ;
        if (tab[i] > *max)
            *max = tab[i] ;
    }
}
```

[3]:
```
int main ()
{
    int tab[5] = { 35, 17, 22, 66, 53 } ;
    int min ;
    int max ;
    minMax(5, tab, &min, &max) ;
    printf("min = %d, max = %d", min, max) ;
}
```

Exécuter la fonction main. Ne pas faire apparaitre le tableau dans les variables du programme: il ne sera pas modifié.

2.3 Graphes de flot de contrôle

Dessiner les graphes de flot de contrôle des programmes suivants.

[4]:
```
int min (int x, int y)
{
  if (x < y)
    return x ;
  else
    return y ;
}
```

[5]:
```
void prog1 ()
{
    int n = 12 ;
    int m = 42 ;
    if (m < n)
        n = 23 ;
    printf("%d",n) ;
}
```

[ ]:
```
double exp (double x, int n)
{
  double result = 1 ;

  for (int i = 0; i < n; i++)
  {
    result = result * x ;
    n++ ;
  }
  return result ;
}
```

[ ]:
```
void triangle (int a, int b, int c)
{
    assert(a >= 0 && b >= 0 && c >= 0) ;
    if (a + b < c || a + c < b || b + c < a)
        printf("Impossible: un triangle verifie l'inegalite triangulaire. \n") ;
    else if (a == 0 && b == 0 && c == 0)
        printf("C'est un point.\n") ;
    else
    {
        int nbEgalite = 0 ;
        if (a == b)
            nbEgalite++ ;
        if (a == c)
            nbEgalite++ ;
        if (b == c)
            nbEgalite++ ;
        if (nbEgalite == 0)
        {
            if (a + b == c || a + c == b || b + c == a)
                printf("C'est un triangle plat.\n") ;
            else
                printf("C'est un triangle scalene. \n") ;
        }
        else if (nbEgalite == 1)
```

```
        {
            if (a + b == c || a + c == b || b + c == a)
                printf("C'est un triangle plat isocele.\n") ;
            else
                printf("C'est un triangle isocele. \n") ;
        }
        else
            printf("C'est un triangle equilateral. \n") ;
    }
}
```

2.4 Design des tests (paragraphe 1.6 du cours 1)

Exercice 1: Pour chacun des programmes de la section 2.3: 1. Etablir une couverture des sommets: proposez des entrées telles que chaque sommet est couvert lors d'une exécution du programme sur l'une des entrées. 2. Etablir une couverture des arcs: même chose avec les arcs. 3. Proposez des tests des cas limites. 4. Proposez des tests qui valident ou invalident les conditions booléennes de plusieurs façons possibles. 5. Proposez des tests dont le format ne correspond pas à celui attendu

Exercice 2: Pour chacun des programmes de la section 2.3: donner un chemin infaisable, ou justifier qu'il n'y en a pas.

2.5 Instructions de modification du graphe de flot de contrôle

2.5.1 Return

1. Simplifier le programme ci-dessous sans changer son comportement.
2. Ecrire les graphes de flot de contrôle des deux programmes.

[ ]:
```
// Teste si syracuse(i) < ceil pour tout i <= N.
bool test (int ceil, int N)
{
    bool b = true ;
    for (int i = 0; i <= N; i++)
    {
        if (syracuse(i) >= ceil)
        {
            b = false ;
            i = N + 1 ;
        }
    }
    return b ;
}
```

2.5.2 Assert

2.5.3 Break

[ ]:
```
bool memTab(int longueur, int tab[longueur], int elt)
{
    bool flag = false ;
    for (int i = 0; i < longueur; i++)
```

```
        if (tab[i] = elt)
        {
            flag = true ;
            break ;
        }
    // il pourrait y avoir des instructions ici
    return flag ;
}
```

1. Dessiner le graphe de flot de contrôle de ce programme.
2. Réécrire ce programme sans l'instruction "break", et en faisant comme si "il y avait des instructions ici". Notamment on ne doit pas "return" dans la boucle.

2.5.4 Continue

[ ]:
```
void sum (int max)
{
    int n ;
    int result  = 0 ;
    printf("Entrez %d entiers pour en faire la somme, les negatifs sont ignores.
↪\n",max) ;
    for (int i = 0; i < max; i++)
    {
        scanf("%d",&n) ;
        if (n < 0)
            continue ;
        result = result + n ;
    }
    printf("La somme totale est de %d",result) ;
}
```

1. Dessiner le graphe de flot de contrôle de ce programme.
2. Réécrire ce programme sans l'instruction "continue".

2.5.5 GOTO (Culture Générale)

2.5.6 Exceptions

Il n'y en a pas en C, mais nous les verrons en OCaml. Intuitivement elles permettent d'ajouter un arc au graphe de flot de contrôle, mais cet arc peut lier deux sommets de fonctions différentes.

2.6 Graphes de flot de contrôle de programmes récursifs

Dessiner les graphes de flot de contrôle des programmes récursifs du TD1 et du programme ci dessous. Appliquer la discipline du 2.4.

[6]:
```
int fact(int n)
{
    if (n==0)
        return 1 ;
    else
        return (fact(n-1)*n) ;
```

```
}
```

2.7 Couverture des sommets VS couverture des arcs

Justifier que si un jeu de test couvre tous les arcs, alors il couvre tous les sommets.

Exhiber un programme dont tous les sommets sont couvrables mais pas tous les arcs.

2.8 Devoirs

A rendre le 16/09: - Sur Caseine, section Interrogations (tout en bas): TODO pour le 16/09. - Sur papier, exécution ligne à ligne de f(12).

[ ]:
```c
int g (int n)
{
    if (n % 4 == 1)
        return 2 ;
    else
        return 1 ;
}

int f (int n)
{
    for (j = 3; j < 6; j++)
    {
        if (j % 3 == 1)
            n = n + g(j) ;
        else
            n = n - 2 ;
    }
    return n ;
}
```