Final Exam
Theoretical Part

EEE243 Applied Computer Programming
13 December 2016, 09:00 – 12:00 hrs

Examiner: Dr. Sidney Givigi, PhD

Instructions:

- **Do not turn this page until instructed to do so.**
- You have 180 minutes to complete the test.
- Questions have the values indicated in the centre column.
- This test has two parts.
    - The *theoretical* portion is closed-book, out of a total of 25 marks.
    - The *practical* portion of the test is open-book (you may have course notes, textbooks and code), and is worth 25 marks.
- You must hand in the *theoretical* exam booklet before you can begin the *practical* part of the test.
- Answer all questions in the test booklet.
- Immediately fill out your name, college number and the number of your workstation on the information card.
- If a question seems unclear, make a *reasonable* assumption, document it, and answer the question as though the assumption were correct. *The examiners will not clarify the meaning of questions during the test*.
- Good luck!

Examen Finale
Partie théorique

GEF243 Programmation informatique appliquée
13 décembre 2016, 09h00–12h00

Examinateur: Capt Adrien Lapointe, MSc

Instructions:

- **Ne tournez pas cette page avant l'instruction de l'examinateur.**
- Vous avez 180 minutes pour compléter le test.
- Les questions ont les valeurs indiquées dans la colonne centrale.
- Ce test a deux parties.
    - La partie *théorique* est à livre fermé et compte pour 25 points.
    - La partie *pratique* est à livre ouvert (vous avez droit à vos notes, à vos livres et code), et compte pour un total de 25 points.
- Vous devez remettre le livret d'examen *théorique* avant de pouvoir commencer la partie *pratique*.
- Répondez à toutes les questions dans le livret d'examen.
- Écrivez immédiatement votre nom, numéro de collège et numéro de station de travail sur la carte d'information.
- Si une question ne vous semble pas claire, faites des suppositions raisonnables, documentez-les et répondez à la question en tenant compte des suppositions. *Les examinateurs ne clarifieront pas le sens des questions pendant le test*.
- Bonne chance!

Beginning of THEORETICAL Portion of Test

1. Explain why two floats cannot be compared with the `==` or `!=` operators.

2. Why the structure below cannot be declared in C?

```
1  struct MyStructure{
2          char name[30];
3          struct MyStructure my_structure;
4  };
```

3. Assuming that you are using the 64 bit version of Windows 10, if one applies the operator `sizeof(Node)` to the type defined in the code below, what would it return? We assume that the compiler allocates 32 bits for a `float`.

```
1  typedef struct NODE_TAG{
2          char last_name[30];
           char first_name[30];
           float average;
3          struct NODE_TAG *p_next;
4  } Node;
```

4. One way of calculating the square root of a positive real number is by using the Newton's Method. This method is used to find the root of any differentiable function and it can be expressed with the following recursive equation:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Where $f'(x_k)$ denotes the derivative of the function $f(.)$ at point x_k. In order to find the square root of a number $a \in \mathbb{R}$, one only needs to use function $f(x)$ and its derivative as:

$$f(x) = x^2 - a$$
$$f'(x) = 2x$$

Notice that the objective is for $f(x) = 0$ when the iterative process is done. It might not always be possible to get to 0 and a stopping value can be specified.

Provide C code for the `square_root(a,tol)`

Début de la partie THÉORIQUE du test

2 1. Expliquez pourquoi il n'est pas possible de comparer deux nombres à point flottant (*float*) avec les opérateurs `==` ou `!=`.

2 2. Pourquoi n'est-il pas possible de déclarer la structure suivante en C?

1 3. En supposant l'utilisation une version 64 bits de Windows 10, quel sera la valeur retournée par l'opération `sizeof(Node)` si appliquée au type définit ci-dessous. On suppose que 32 bits sont alloués à un `float`.

8 4. Une façon de calculer la racine carrée d'un nombre réel positif est d'utiliser la méthode de Newton. Cette méthode est utilisée pour trouver la racine de n'importe quelle fonction différentiable et peut être exprimée par l'équation récursive suivante :

Où $f'(x_k)$ dénote la dérivée de la fonction $f(.)$ au point x_k. Afin de trouver la racine carrée d'un nombre $a \in \mathbb{R}$, nous n'avons qu'à utiliser la fonction $f(x)$ et ses dérivées comme :

Notez que l'objectif est d'avoir $f(x) = 0$ lorsque le processus itératif est terminé. Toutefois, dans certains cas, il ne sera pas possible d'atteindre 0 et une valeur d'arrêt peut être spécifiée.

Donnez le code C pour la fonction

function, where the square root of `a` is calculated to a given tolerance `tol`.

The prototype of the function is:

`square_root(a,tol)` où la racine carrée de `a` est calculée à une tolérance `tol`.

Le prototype de la fonction est :

```
float square_root(float a, float tol);
```

5. Consider the following code. **9** 5. Considérez le code suivant.

```
#include <stdio.h>
#include <stdlib.h>

typedef int (*INT_FUNCTION) (int);

int square(int i) {return i*i;}
int cube(int i) {return i*i*i;}
int factorial(int n) {
	int temp = 1;
	for (int i = 0; i <= n; i++){
		temp *= i;
	}
	return temp;
}

int *apply(int *array, int size, INT_FUNCTION fun){
	static int total = 0;
	int *output = (int*)malloc(size*sizeof(int));
	if (output != NULL){
		for (int i = 0; i < size; i++){
			output[i] = fun(array[i]);
			total+=output[i];
		}
		printf("\n\nTotal is %d\n\n", total);
	}
	return output;
}

void print_array(int* array, int size){
	if (array != NULL){
		for (int i = 0; i < 5; i++){
			printf("%d\t",array[i]);
		}
		printf("\n\n");
	} else {
		printf("\nArray is invalid\n\n");
	}
}
```

```
38
39  int choose_function(){
40          int option;
41          printf("Choose the function you would like to run:\n \t");
42          printf(" 1 - Square\n \t 2 - Cube\n \t 3 - Factorial\n");
43          scanf(" %d",&option);
44          return &option;
45  }
46
47  int main(void){
48          INT_FUNCTION fn_arr[3] = {square, cube, factorial};
49          int array[5] = {1, 2, 3, 4, 5};
50
51          for (int i = 0; i < 2; i++){
52                  print_array(apply(array, 5, fn_arr[choose_function() - 1]),5);
53          }
54
55          return 0;
56  }
```

This program offers three choices of operations over integers to the user: calculate the square, the cube and the factorial of numbers. Answer the following questions:

a. There are two errors in the code, one of logic and one of syntax. Identify both.

b. Where variables `temp` (line 9), `total` (line 17), `output` (line 18) and `array` (line 49) are located in memory?

c. After fixing the errors, assume that the program is executed twice and the user chooses inputs 2 and 1. Give the output of the program as it is printed in line 24 and line 31.

6. Explain the difference between `malloc()` and `calloc()` and give code examples for the use of both.

Cette fonction offre à l'utilisateur trois choix d'opérations sur des entiers : calculer le carré, le cube et la factoriel de nombres. Répondez aux questions suivantes :

(2) a. Il y a deux erreurs dans le code : une de logique et une de syntaxe. Identifiez-les.

(3) b. Où sont les variables `temp` (ligne 9), `total` (ligne 17), `output` (ligne 18) et `array` (ligne 49) situées en mémoire?

(4) c. Après avoir corrigé les erreurs, supposez que le programme est exécuté deux fois et que l'utilisateur choisi les entrées 2 et 1. Donnez la sortie du programme qui est affichée aux lignes 24 et 31.

3 6. Expliquez la différence entre `malloc()` et `calloc()`. Donnez des exemples de code de l'utilisation de chacun.

End of THEORETICAL Portion of Test

Fin de la partie THÉORIQUE du test