Question 1

- ◊ Le langage C est un langage évolué
- ◊ Le Langage C est un langage dit interprété
- ◊ Le langage C est un langage dit portable
- ◊ Le langage C est un langage dit d'assemblage

Question 2

Les noms de versions suivants du C sont corrects

- ◊ C 99
- ◊ C 11
- ◊ C K&R
- ◊ C 89 et C 90

Question 3

En langage C

- ◊ ***const int n=10 ;*** est une déclaration et non une définition
- ◊ ***typedef struct personne pers*** est une définition
- ◊ ***int f(int x);*** est une définition
- ◊ ***define V(p) 4*(p)*** est une définition

Question 4

En langage C, **#include <stdio.h>**

- ◊ est une expression
- ◊ Est une instruction
- ◊ Peut être placé n'importe où dans le code

Question 5

En langage C, **#include « fic.h »**

- ◊ est une expression
- ◊ Est une instruction
- ◊ Peut être placé n'importe où dans le code
- ◊ Est incorrect

Question 6

En langage C,

- ◊ Un identificateur est utilisé pour nommer une variable, un type, une fonction, un programme
- ◊ La longueur d'un identificateur ne doit pas dépasser 8 caractères
- ◊ **restrict** est un identificateur valide
- ◊ Les seules portées possibles sont **Fonction, Bloc, Fichier**

Question 7
En langage C, le type permet
- ◊ De savoir la portée d'une variable
- ◊ De savoir la classe d'allocation d'une variable
- ◊ De savoir comment stocker une fonction en mémoire
- ◊ De savoir la taille mémoire nécessaire au stockage d'une fonction en mémoire

Question 8
En langage C,
- ◊ A chacun des 5 types signés on a associé un type non signé de taille supérieur désigné par le préfixe **unsigned** après le spécificateur de type
- ◊ A chacun des 5 types signés on a associé un type non signé de taille inférieur désigné par le préfixe **unsigned** après le spécificateur de type
- ◊ A chacun des 5 types signés on a associé un type non signé de même taille désigné par le préfixe **unsigned** après le spécificateur de type

Question 9
En langage C, le standard fixe la taille mémoire des types entiers comme suit :
- ◊ short = 2 octets
- ◊ int = 4 octets
- ◊ long long = 8 octets
- ◊ Les valeurs extêmes sont définies dans **limits.h** pour une implémentation donnée

Question 10
En langage C, les types flottants sont au nombre de :
- ◊ 4
- ◊ 2
- ◊ 3

Question 11
En langage C, l'expression **4.0 / 2** :
- ◊ N'est pas acceptée par le compilateur
- ◊ Est équivalente à 4.0 / 2.0
- ◊ Est équivalente à 4 / 2
- ◊ Est équivalente à 4 / 2.0

Question 12
En langage C, **sizeof(x)** :
- ◊ Est un appel de fonction
- ◊ Retourne une valeur de type ptr_diff
- ◊ Retourne une valeur de type size_t

Question 13
En langage C, **char*** t[10] :
- ◊ Est une chaine de caractères
- ◊ Est un tableau de caractères
- ◊ Est un pointeur vers un tableau de 10 caractères

Question 14

En langage C, **int* (*t)[10]** :

- ◊ Est une tableau de 10 pointeurs vers int
- ◊ Est un tableau de 10 pointeur de pointeur vers int
- ◊ Est un pointeur vers un tableau de 10 pointeurs vers int

Question 15

En langage C, **int* f(){...}** :

- ◊ Est incorrecte car une fonction ne renvoie par un pointeur
- ◊ Est incorrecte car une fonction ne renvoie par un tableau
- ◊ Est une fonction qui peut renvoyer un tableau d'entier

Question 16

En langage C, **int f(int t[10]) {...}** :

- ◊ Est une fonction qui peut renvoyer un tableau d'entier
- ◊ Est incorrecte car une fonction ne prend pas de tableau en paramètre
- ◊ Est incorrecte à cause de la taille du tableau

Question 17

En langage C, **union nombre {long n ; int x ;}** :

- ◊ Fait que les champ n et x soient de même taille
- ◊ Fait que les champ n et x soient de même type
- ◊ Fait que les champ n et x soient exactement les mêmes en mémoire

Question 18

En langage C, **struct{unsigned int b0_2 :3; signed int b3_7 :5; unsigned int :7; unsigned int b15 :1; }deuxOctets;**

- ◊ N'est pas correct
- ◊ N'est pas correct à cause de **unsigned int :7**
- ◊ Est correct et définit une variable deuxOctets qui tient sur 4*siseof(int) octets au moins
- ◊ Est correct et définit une variable deuxOctets qui peut tenir sur siseof(int) octets

Question 19

En langage C, **int t[6]= {1,2,3,4,5,6}, *p1= t, *p2 = &t[5];**

- ◊ N'est pas correct
- ◊ Est correct , mais p2-t ne l'est pas
- ◊ Est correct et fait que p2-p1 soit égal à 5*sizeof(int)

Question 20

En langage C, **struct t{int n; struct t* suiv; };**

- ◊ N'est pas correct car une structure ne peut avoir un champ de son type
- ◊ Permet de représenter un tableau d'entiers
- ◊ Permet de définir une liste d'entiers

SN/UCAD/ESP/DGI/L3 (DIC1 INFO et DIC1 TR)
Interrogation écrite n° 1 de Programmation C

Prénom et nom: .....Classe:.....

Question 1

En C, les conversions de type

- Sont dites explicites lorsqu'elles sont faites automatiquement par le compilateur
- Sont dites implicites lorsqu'elles sont faites sur demande du programmeur
- Permettent de retourner la valeur d'une expression dans un type différent du sien
- Permettent de retourner la valeur d'une variable dans un type différent du sien

Question 2

En C, la promotion entière (également appelée promotion numérique)

- S'applique lorsqu'une expression arithmétique contient un opérande de type int
- Est réalisée de façon systématique par le compilateur
- Est réalisée par le compilateur après la conversion implicite
- Est réalisée de facon automatique par le compilateur

Question 3

En C, les conversions arithmétiques implicites

- Ne s'effectuent pas dans le cas de tous les opérateurs arithmétiques binaires
- Permettent au programmeur d'écrire des expressions mixtes
- Sont effectuées lorsque les opérandes d'un opérateur arithmétique binaire sont de types différents

Question 4

En C

- Une déclaration permet toujours d'associer un type à une variable
- Une déclaration est une définition qui alloue de l'espace mémoire
- Une déclaration fixe l'interprétation et les propriétés d'un ou de plusieurs identificateurs en leur associant un type

Question 5

En C

- Seules les variables de certains types arithmétiques doivent être déclarées avant d'être utilisées
- A cause de la promotion entière, les variables de types char et short peuvent ne pas être déclarées avant d'être utilisées
- Les déclarations peuvent se situer à n'importe quel niveau d'un bloc

SN/UCAD/ESP/DGI/L3 (DIC1 INFO et DIC1 TR)
Interrogation écrite n° 3 de Programmation C

Prénom et nom: .....Classe:.....

Question 1
Les définitions de constantes longues entières suivantes sont correctes

- ◊ 0X31Gll
- ◊ 0394ul
- ◊ 0x1faul
- ◊ 10

Question 2
Les identificateurs suivants sont incorrects (ne peuvent être utilisés par un programmeur) en C

- ◊ _Bool
- ◊ unsigned
- ◊ longlong
- ◊ trigraphe

Question 3
En langage C

- ◊ ***const int N=10 ;*** est une déclaration et non une définition
- ◊ ***int f();*** est une définition
- ◊ ***define N 10*** est une définition
- ◊ L'instruction ***n=x+1 ;*** ne compile pas si n est un int et x un float

Question 4
En langage C, **f(x)**

- ◊ Est une expression
- ◊ Est une instruction
- ◊ Est une instruction-expression

Question 4
En langage C

- ◊ La classe de mémorisation par défaut pour les variables est **extern**
- ◊ La classe de mémorisation **register** n'est pas définie pour les fonctions et permet de déclarer des variables globale.
- ◊ *int t []={1,2,3}* est erronée comme *char ch[5]="DIC1"* ;

SN/UCAD/ESP/DGI/L3 (DIC1 INFO et DIC1 TR)
Interrogation écrite n° 3 de Programmation C

Prénom et nom: .....Classe:.....

Question 1
char* p[10]
◊ p est un pointeur vers un tableau de char *
◊ p est un pointeur vers un tableau de chaines de caractères
◊ p est un pointeur vers un tableau 10 chaines de caractères

Question 2
char (*p) [10]
◊ p est un pointeur vers un tableau de char *
◊ p est un pointeur vers un tableau de chaines de caractères
◊ p est un pointeur vers un tableau de 10 char *

Question 3
printf() et scanf()
◊ renvoient le nombre de caratères écris ou lus
◊ prennent des arguments de type cheine de caractères
X ◊ sont des fonctions à arguments en nombre variables

Question 4
enum {noir, blanc, bleu, vert, rouge};
◊ Est incorrect
◊ Est correct et fait que noir=blanc=bleu=vert=rouge=1
◊ Est correct et fait que noir=1, blanc=2, bleu=3, vert=4, et rouge=5

Question 4
En langage C, soit p un tableau de taille 10
◊ p[-1] ne passe pas la compilation
◊ p[x] ne passe pas la compilation si x est supérieur à 9
◊ p=p+1 ajoute 2 à l'adresse p