QCM sur le langage C

Enseignants CSC4002

CSC 4002

Télécom SudParis Octobre 2015

1 QCM « syntaxe et fonctions en langage C »

Instructions. Pour chacune des questions, plusieurs réponses peuvent être correctes, vous pouvez donc cocher plusieurs cases !

1. Une fonction en langage C 89 est définie par :

   faux son nom et le type de valeur retournée.

   Justification : *Bien que l'édition de liens n'utilise que le nom pour trouver la fonction, le type de retour ainsi que les types des arguments sont nécessaires pour la cohérence entre les arguments passés et l'utilisation de la valeur retournée.*

   faux son nom seul.

   Justification : *Bien que l'édition de liens n'utilise que le nom pour trouver la fonction, le type de retour ainsi que les types des arguments sont nécessaires pour la cohérence entre les arguments passés et l'utilisation de la valeur retournée.*

   vrai le type de retour, le nom, le nombre et les types d'arguments formels de la fonction.

   Justification : *En langage C, les arguments font partie du prototype. Il n'existe cependant pas de possibilité d'avoir deux fonctions qui portent le même nom et qui ont des paramètres différents en nombre ou en type.*

2. Le langage C réalise-t-il le passage d'arguments par valeur ou par adresse ?

   faux Par valeur pour les constantes et par adresse pour les variables.

   Justification : *Il n'y a pas de différence de traitement entre les variables et les constantes.*

   vrai Par adresse pour les pointeurs et les tableaux, par valeur pour les variables et les constantes.

   Justification : *Il n'y a pas de différence de traitement entre les adresses et les autres types. L'ensemble des paramètres est passé par copie de la valeur du paramètre dans une variable. Ainsi, une constante entière devient une variable entière et une adresse d'entier (constante du type adresse d'entier) devient un pointeur d'entier.*

   faux Par valeur quel que soit le type d'arguments à la fonction appelée.

   Justification : *Non les tableaux et les pointeurs sont passés par adresse.*

3. En langage C, le retour de fonction :

   vrai Doit être d'un type cohérent avec le type du retour de la fonction

   Justification : *Oui sinon il faut réaliser un 'cast'.*

   vrai Ne peut retourner qu'un résultat à la fois

   Justification : *Oui, même s'il y a plusieurs appels à 'return' dans le code de la fonction, un seul d'entre eux est appelé et il ne tranportera qu'une seule valeur.*

   vrai Peut retourner une structure

   Justification : *Oui, c'est la seule solution pour retourner plusieurs valeurs calculées par une fonction.*

   faux Peut retourner le contenu d'un tableau .

   Justification : *Non, une fonction ne peut pas retourner le contenu d'un tableau, à moins d'intégrer ce contenu dans une structure.*

4. Une fonction qui retourne un pointeur peut :

   faux Retourner un pointeur sur n'importe quelle variable locale.

   Justification : *Non, la variable locale doit être rémanente pour pouvoir être utilisée lorsque le fonction est terminée (ajouter 'static' sur la ligne de définition).*

   vrai Retourner un pointeur sur une zone mémoire correspondant aux variables globales.

   Justification : *Oui, c'est sans danger.*

   vrai Retourner un pointeur sur une zone mémoire allouée dynamiquement

Justification : *Oui, c'est un peu dangereux car il est facile de déborder de cette zone.*

5. Les types suivants existent en langage C 89 :

   vrai caractère

   Justification : *Le type caractère est appelé 'char'. Il correspond à un type entier exprimé sur huit bits.*

   vrai entier

   Justification : *Il existe plusieurs types entier, appelés 'short', 'int' et 'long'. Leurs tailles sont exprimées selon la relation suivante : sizeof(short) <= sizef(int) <= sizeof(long).*

   vrai virgule flottante

   Justification : *Il existe plusieurs types permettant de réaliser des opérations en virgule flottante : 'float', 'double' et 'long double'.*

   faux booléen

   Justification : *Le type 'booléen' n'existe pas en langage C 89.*

6. L'opérateur d'affectation en langage est un opérateur sans statut particulier. Il est ainsi possible de réutiliser la constante fournie par une affectation. Par exemple, 'a = b = c = 1 ;' affecte la valeur 1 aux variables c, b et a.

   faux Non, l'opérateur d'affectation est un opérateur spécialisé.

   Justification : *Contrairement à ses prédécesseurs, le langage C n'attribue pas de statut particulier à cet opérateur.*

   vrai Oui, l'opérateur d'affectation est un opérateur banalisé.

   Justification : *Contrairement à ses prédécesseurs, le langage C n'attribue pas de statut particulier à cet opérateur.*

7. En langage C, l'instruction composée est réalisée à partir d'instructions groupées dans un bloc entre deux accolades.

   vrai Oui

   Justification : *En effet, une instruction en langage C peut être une instruction simple ou un bloc d'instructions.*

   faux Non

   Justification : *En effet, une instruction en langage C peut être une instruction simple ou un bloc d'instructions.*

8. Une instruction simple est toujours terminée par un point virgule.

   vrai Oui

   Justification : *En effet, c'est une expression terminée par un ';'.*

   faux Non

   Justification : *Si, une instruction simple est une expression terminée par un ';'.*

9. En langage C, les instructions de contrôle permettant de faire des choix sont les suivantes :

   vrai `if (expression) instruction else instruction`
   où la partie else est facultative.

   Justification : *C'est exactement la bonne forme. Les parenthèses autour de l'expression sont obligatoires. L'instruction est définie comme nous l'avons vu dans la question précédente, c'est-à-dire soit une instruction simple soit un bloc d'instructions.*

   faux `if expression then instruction else instruction`

   Justification : *Les parenthèses autour de l'expression sont obligatoires et il n'y a pas de then. C'est en 'shell-script' qu'il y a un 'then'.*

vrai
```
switch (expression) {
 case const1 :
       instructions
...
 case constn :
       instructions
}
```

Justification : *Les tables de sauts sont structurées comme des ensembles d'instruction avec des points d'entrée caractérisés par des valeurs de constantes entières.*

faux
```
case expression in
  const1 :
       instructions
...
 constn :
       instructions
esac
```

Justification : *Cette syntaxe correspond encore à du shell-script.*

10. En langage C, les instructions de contrôle permettant de faire des boucles sont les suivantes :

faux `for (instruction; instruction; instruction) instruction`

Justification : *Non, sinon il serait possible de mettre des blocs d'instructions entre les parenthèses.*

vrai `for (expression;expression;expression) instruction`

Justification : *C'est exactement la bonne forme. La première expression est une expression d'initialisation, la seconde expression correspond à la condition de passage dans la boucle et la dernière expression est le plus souvent utilisée pour faire évoluer la boucle.*

vrai `while (expression) instruction`

Justification : *C'est la bonne forme. L'expression correspond à la condition de passage dans la boucle.*

vrai `do instruction while (expression)`

Justification : *C'est la bonne forme. L'expression correspond à la condition de passage dans la boucle. Cette forme de boucle garantit que l'instruction est réalisée au moins une fois.*