+69/1/56+

# ESIR SYS1 CC2 2021–2022
**20 janvier 2022**

Nom et Prénom

**X X**
**numéro d'étudiant : 99999990**

Durée : 1h, Notation : sur 20 points

---

## 1 Première partie : QCM (10 points)

**Instructions :**

**Cochez** clairement la case de la réponse que vous pensez être juste. Il y a **une seule réponse** juste par question.

**Barème :**

+0.5 pour chaque réponse correcte
$-0.5/m$ pour chaque réponse fausse (où m+1 est le nombre de réponses possibles)

**Question 1** Laquelle des assertions suivantes est-elle vraie ?

- ☐ Un cache L1 est plus rapide qu'un cache L2.
- ☐ Un cache L2 est plus rapide qu'un cache L1.

**Question 2** Comment réserver 16 octets (`10h` en hexadécimal) sur la pile en assembleur x86 64 bits ?

- ☐ `add rbp,10h`
- ☐ `add rsp,10h`
- ☐ `sub rbp,10h`
- ☐ `sub rsp,10h`

**Question 3** Soit le code suivant

```
#include <stdio.h>

void foo(int* i) {
  *i = 10;
}
int main( int argc, char** argv) {
  int i = 5;
  foo(&i);
  printf("%i\n",i);
}
```

Quelle valeur ce code imprime-t-il sur la console ?

- ☐ 5
- ☐ 10
- ☐ le contenu de la mémoire à l'adresse 10

**Question 4** En C, combien d'arguments prend la fonction `fprintf` ?

- ☐ aucun
- ☐ toujours 1
- ☐ toujours 2
- ☐ 2 ou plus selon les cas

**Question 5** Sur combien de d'octets fonctionne le registre `rax` dans les architectures x86-64 ?

- ☐ 4 octets
- ☐ 8 octets
- ☐ 16 octets

**Question 6** Un exécutable peut-il ne contenir qu'une partie du code binaire nécessaire à son exécution ?

- ☐ non
- ☐ oui

**Question 7** Sur combien de bits fonctionne le registre `ecx` dans les architectures x86-64 ?

- ☐ 16
- ☐ 32
- ☐ 64

**Question 8** Si l'exécutable `a.out` utilise la librairie partagée (*shared library*) `libfoo.so`, trouvera–t–on le contenu de `libfoo.so` dans le fichier `a.out` ?

- ☐ non
- ☐ oui

**Question 9** On considère l'extrait de code en C suivant :

```
int16_t i   = 10;
char* ptr_i = (char*)&i;
ptr_i[0]    = 2 ;
printf("%i\n", (int)i );
```

La ligne `printf("%i\n", (int)i );` permet d'imprimer un entier. Quelle sortie produit cet extrait de programme lorsqu'il s'exécute sur un processeur Intel de la famille x86 ?

- ☐ 2
- ☐ 10
- ☐ 256
- ☐ 522

**Question 10** Soit le code assembleur suivant

```
mov  rax, 4
mov  rbx, 5
push rax
push rbx
mov  rbx, 3
mov  rax, 4
pop  rbx
pop  rax
```

À la suite de ce code :

- ☐ `rax` et `rbx` contiennent 2 et 3 respectivement.
- ☐ `rax` et `rbx` contiennent 5 et 4 respectivement.
- ☐ `rax` et `rbx` contiennent 4 et 5 respectivement.
- ☐ `rax` et `rbx` contiennent 3 et 2 respectivement.



**Question 11** Sur un processeur Intel, le passage de paramètre par valeur est–il compatible avec un passage de paramètre par la pile ?

- ☐ oui
- ☐ non

**Question 12** Dans le code C suivant, la ligne "`ptr_i[0] = 2 ;`" est-elle syntaxiquement correcte ?

```
int main( int argc, char** argv) {
  int array[] = {1,2,3} ;
  int* ptr_i  = array ;
  ptr_i[0]    = 2 ;
} // EndMain
```

- ☐ oui, la ligne "`ptr_i[0] = 2 ;`" est syntaxiquement correcte.
- ☐ non, la ligne "`ptr_i[0] = 2 ;`" n'est pas syntaxiquement correcte.

**Question 13** Comment peut–on compiler le fichier `test.c` avec `gcc` sans réaliser l'édition de lien (*linking*) ?

- ☐ `gcc -c test.c`
- ☐ `gcc test.c`
- ☐ `gcc -o test.c`

**Question 14** Quel est l'ordre de grandeur de la taille d'un cache de niveau 1 (L1) sur un processeur moderne ?

- ☐ 1 megaoctet
- ☐ 64 kilooctets
- ☐ 10 octets
- ☐ 1 gigaoctet

**Question 15** On considère l'extrait de code en C suivant :

```
char x = 'H';
x--;
printf("%c\n", x );
```

La ligne `printf("%c\n", x );` permet d'imprimer un caractère. Quelle sortie produit cet extrait de programme ?

- ☐ Aucune sortie, le programme ne compile pas.
- ☐ H
- ☐ G

**Question 16** Si deux programmes `A` et `B` sont compilés avec la même librairie dynamique `L`, le code de la librairie `L` se retrouve–t–il dupliqué dans chacun des fichiers exécutables de `A` et de `B` ?

- ☐ oui
- ☐ non

**Question 17** Est–il possible d'effectuer des opération arithmétiques (additions, soustractions) sur un pointeur en C :

- ☐ non
- ☐ oui

**Question 18** Soit le code suivant

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

void foo(char** c) {
  char a_string[] = "SYS1 c'est top!";
  *c = malloc(strlen(a_string)+1);
  strcpy(*c,a_string);
}
int main(int argc, char** argv) {
  char* c;
  printf("%s\n",c);
}
```

Dans quelle région mémoire la chaîne contenue dans `a_string[]` est-elle stockée ?

- ☐ sur le tas
- ☐ sur la pile
- ☐ dans le segment de texte
- ☐ dans le segment BSS

**Question 19** Dans la ligne C suivante, combien d'octets sont-ils réservés en mémoire pour stocker la chaîne `"abcd"` ?

```
char* my_string="abcd" ;
```

- ☐ 2 octets
- ☐ 4 octets
- ☐ 5 octets
- ☐ cela dépend

**Question 20** On considère le code assembleur suivant pour implémenter une fonction `foo` qui ajoute 33 au paramètre qui lui est passé en entrée :

```
foo:  add rax,33
      ret
```

Quel type de passage de paramètre(s) cette fonction `foo` utilise–t–elle ?

- ☐ le passage de paramètre(s) par la pile
- ☐ le passage de paramètre(s) par registre

## 2 Deuxième partie : Questions ouvertes et problèmes (10 points)

**Question 21** L'on souhaite imprimer 5 fois de suite la chaîne de caractère '`In the loop!`' en utilisant une boucle en assembleur. Un premier essai livre le code assembleur suivant : (Seules les parties du code pertinentes sont montrées.)

```
SECTION     .data
message:    db "In the loop!", 10      ; note the newline at the end
msgLen:     equ $-message

SECTION    .text
  [..]
  mov       rax, 0

begin:
  mov       rax, 1              ; system call for write
  mov       rdi, 1              ; file handle 1 is stdout
  mov       rsi, message        ; address of string to output
  mov       rdx, msgLen         ; number of bytes
  syscall                       ; invoke operating system to do the write

  inc       rax
  cmp       rax, 5
  jl        begin
  [..]
```

Ce code compile, mais l'exécution n'imprime qu'une seule fois le message :

```
$ ./loop_buggy
In the loop!
$
```

1. Pourquoi observe–on ce comportement ?
2. Comment corrigeriez–vous le code pour que la boucle s'exécute bien 5 fois. Vous proposerez *deux* solutions : l'une utilisant une variable supplémentaire dans la section `.data`, et l'autre s'appuyant sur la pile. Dans les deux cas vous prendrez soin de bien expliquer votre solution.

**[5 points]** ☐A+ ☐A ☐B ☐C ☐E ☐F

**Question 22** On considère le code C suivant :

```
#include <stdio.h>

int main( int argc, char* argv[]) {

  if (argc==1) return 1;

  int i=0 ;
  char my_string[5];

  while (argv[1][i]!=0) {
    my_string[i]=argv[1][i];
    printf("Copying byte n.%i: %c\n",i,argv[1][i]);
    i++;
  }
  my_string[i]=0; // copying trailing \0

  printf("%s\n",my_string); // safer

} // EndMain
```

Lorsqu'on le compile avec `gcc` puis qu'on l'exécute on obtient la sortie suivante

```
$ gcc buggy_string_exam.c
$ ./a.out "HELLO,"
Copying byte n.0: H
Copying byte n.1: E
Copying byte n.2: L
Copying byte n.3: L
Copying byte n.4: O
Copying byte n.44: s
Copying byte n.45: s
Copying byte n.46: h
HELLO/
$
```

— Ce code contient un bug. En vous appuyant sur la sortie indiquée, expliquez d'où provient ce bug, et proposez une ou des hypothèses pour expliquer pourquoi la valeur de l'index `i` saute de `4` à `44` dans la sortie imprimée sur la console. `[3 points]`

— Comment corrigeriez–vous ce bug ? Il n'est pas nécessaire de proposer un code corrigé ici : il suffit d'expliquer quelle stratégie pourrait selon vous éviter ce problème. (Plusieurs possibilités existent, mais il suffit d'en expliquer une seule pour répondre.) `[2 points]`

**[5 points]** ☐A+ ☐A ☐B ☐C ☐E ☐F