Université libre de Bruxelles

Faculté des Sciences
Départment d'informatique

Langages de programmation

(INFO-F105)

Première année du programme de bachelier en Science Informatique

Examen de juin 2021

- L'examen comprend quatre questions, chacune notée sur 10 points
- La pondération associée à chaque sous-question est fournie
- Répondez à chaque question sur une feuille séparée fournie à cet effet
- Indiquez vos nom, prénom et numéro d'étudiant sur chaque feuille de réponse
- Toutes les notes de cours, transparents et syllabus sont autorisés
- Calculatrices, ordinateurs, téléphones interdit
- Durée de l'examen : 2:00
- Pour rappel, la note de cours est consituée de soit
  - 2/3 examen + 1/3 projet
  - 100% examen

  (la note la plus favorable des deux)

Question 1

Observez le code en assembleur ci-dessous, compilé pour une architecture 32bits.

```
1  %include "io.inc"
2  SECTION .data
3  ;Les 10 nombres sont des "words", donc sur 16bits.
4  nombres DW 314, 399, 490, 98, 143, 379, 494, 437, 256, 435
5
6  section .text
7  global CMAIN
8  CMAIN:
9      MOV AX, [nombres]
10     MOV BX, AX
11     MOV ECX, 9
12 etiquette0:
13     MOV DX, [nombres + 2 * ECX]
14     CMP AX, DX
15     JGE etiquette1
16     MOV AX, DX
17 etiquette1:
18     CMP BX, DX
19     JLE etiquette2
20     MOV BX, DX
21 etiquette2:
22     DEC ECX
23     JNZ etiquette0
24
25     PRINT_UDEC 2, AX
26     NEWLINE
27     PRINT_UDEC 2, BX
28     NEWLINE
29
30     XOR EAX, EAX
31     RET
```

1. Expliquez en français ce que fait ce code (pas de description ligne par ligne !). Qu'affichent les deux instructions `PRINT_UDEC` vers la fin du programme? **[3 points]**

2. Observez ce second code qui effectue le même traitement mais d'une manière légèrement différente. Expliquez la différence avec la première méthode. Quel en est l'inconvénient ?
[3 points]

```
1  %include "io.inc"
2  SECTION .data
3  nombres DW 314, 399, 490, 98, 143, 379, 494, 437, 256, 435
4
5  section .text
6  global CMAIN
7  CMAIN:
8      MOV AX, [nombres]
9      MOV BX, [nombres]
10     MOV ECX, 9
11 etiquette0:
12     CMP AX, [nombres + 2 * ECX]
13     JGE etiquette1
14     MOV AX, [nombres + 2 * ECX]
15 etiquette1:
16     CMP BX, [nombres + 2 * ECX]
17     JLE etiquette2
18     MOV BX, [nombres + 2 * ECX]
19 etiquette2:
20     DEC ECX
21     JNZ etiquette0
22
23     PRINT_UDEC 2, AX
24     NEWLINE
25     PRINT_UDEC 2, BX
26     NEWLINE
27
28     XOR EAX, EAX
29     RET
```

3. Sans inclure de librairie additionnelle à ce qui est déjà présent sur la page de réponse, écrivez une fonction appelée `fonction_mystere` en C++ qui effectue le même traitement (le type `short` est un entier sur 16 bits). Vous trouverez un squelette de code sur la page de réponse. **[4 points]**

Question 2

Étant donné le code C++ à la page suivante, répondez **brièvement** (2-3 lignes maximum) aux questions suivantes :

1. La fonction `main` est définie à la fin du fichier. Est-il possible de la placer avant les autres fonctions ? Si oui : expliquez comment. Si non : justifiez pourquoi. **[1 point]**

2. Expliquez les **différences** entre les **modes de passage de paramètres** entre les fonctions `fonction1`, `fonction2` et `fonction3`. **[1 point]**

3. Ces trois fonctions ont-elles un effet de bord ? Justifiez. **[1 point]**

4. Les différents modes de passage des paramètres de ces trois fonctions peuvent-ils avoir un impact sur la sécurité du programme ? Justifiez. **[1 point]**

5. Les différents modes de passage des paramètres de ces trois fonctions peuvent-ils avoir un impact sur l'efficacité du programme ? Justifiez. **[1 point]**

6. Que sera-t-il affiché par le programme lors de l'exécution des lignes 42 et 43 (`fonction1`) ? **[1 point]**

7. Que sera-t-il affiché par le programme lors de l'exécution des lignes 46 et 47 (`fonction2`) ? **[1 point]**

8. Que sera-t-il affiché par le programme lors de l'exécution des lignes 51 et 52 (`fonction3`) ? **[1 point]**

9. Expliquez l'utilité du mot-clé `inline` à la ligne 4. **[1 point]**

10. Quel est le mécanisme permettant de créer deux fonctions ayant le même nom (exemple : les fonctions `afficher_variables`) ? Donnez un exemple d'instruction permettant d'appeler chacune de ces fonctions. **[1 point]**

```
1  #include <iostream>
2  using namespace std;
3
4  inline void afficher_variables(int a,  int b, int c) {
5      cout << "a=" << a << ", b=" << b << ", c=" << c << endl;
6  }
7
8  inline void afficher_variables(int *a, int *b, int *c) {
9      cout << "a=" << *a << ", b=" << *b
10          << ", c=" << *c << endl;
11 }
12
13 void fonction1(int a, int b, int c) {
14     if(a < b) {
15         int a = 30;
16         int b = a - c;
17         c = 20;
18         afficher_variables(a, b, c);
19     }
20     afficher_variables(a, b, c);
21 }
22
23 void fonction2(int &a, int &b, int &c) {
24     if(a < b) {
25         a = b + c;
26         int b = 80;
27         afficher_variables(a, b, c);
28     }
29     afficher_variables(a, b, c);
30 }
31
32 void fonction3(int *a, int *b, int *c) {
33     if(*a < *b) {
34         *a = 0;
35         afficher_variables(*a, *b, *c);
36     }
37     afficher_variables(*a, *b, *c);
38 }
39
40 int main() {
41     int a = 10, b = 20, c = 30;
42     fonction1(a, b, c);
43     afficher_variables(a, b, c);
44
45     a = 10, b = 20, c = 30;
```

```
46      fonction2(a, b, c);
47      afficher_variables(a, b, c);
48
49      a = 10, b = 20, c = 30;
50      int *pa = &a, *pb = &b, *pc = &c;
51      fonction3(pa, pb, pc);
52      afficher_variables(a, b, c);
53
54      return 0;
55  }
```

Question 3

Soit

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

avec $n \in \mathbb{N}^+, k \in \mathbb{N}^+, n \geq k$. Rappelons que : $i! = i \times (i-1) \times (i-2) \times \cdots \times 1$

1. Implémentez une classe **CoefB** avec:
   - un ou plusieurs constructeurs (par défaut $n = 1$ et $k = 1$)
   - un destructeur
   - les getters
   - les setters (sans tests)
   - une méthode entière non-signée *value* qui renvoie la valeur de $\binom{n}{k}$ en utilisant la méthode ci-dessus si $n \geq k$ sinon la méthode renvoie 0. Vous pouvez utiliser la fonction `fact` qui renvoie la factorielle de n:

```
1 unsigned fact(const unsigned &n) {
2     return (n > 1) ? n*fact(n-1) : 1;
3 }
```

[7 points]

2. Surchargez l'opérateur << sur `std::ostream` pour pouvoir faire :

   ```
   cout << cb << endl;
   ```

   avec `cb` une instance de **CoefB**. Si `cb` vaut $\binom{4}{2}$, l'output affichera :

   ```
   (4,2) = 6
   ```

[3 points]

Remarque : veillez à utiliser judicieusement les mots clés vus tels que **public**, **private**, **const** et les références.

Question 4

1. Considérez le code C++ suivant:

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int secretTab[5] = {2,3,7,19,82};
6      int publicTab1[4] = {2,4,6,8};
7      int secretValue = 357;
8      float publicTab2[3] = {0.7,15,0.5};
9
10     int a = *((&publicTab1[2])+1)+5;
11     //float b = publicTab2[10];
12     //int c = publicTab2[10];
13
14     cout << "printing publicTab1: ";
15     for(int i=0; i<=4; i=i+1)
16         cout << publicTab1[i] << " ";
17
18     return 0;
19 }
```

(a) Quelle est la valeur de la variable a après la ligne 10 ? Justifiez votre réponse. **[1 point]**

(b) Si on décommente la ligne 11, le programme produira-t-il une erreur à la compilation ? Justifiez votre réponse. **[1 point]**

(c) Si on décommente la ligne 12, le programme produira-t-il une erreur à la compilation ? Justifiez votre réponse. **[1 point]**

(d) Après compilation et exécution du code on observe le résultat suivant

```
1 printing publicTab1: 2 4 6 8 2
```

Expliquez le résultat obtenu. **[2 points]**

(e) Est-il possible d'obtenir le même genre de résultat en Java ? Justifiez votre réponse. **[1 point]**

(f) On peut s'étonner qu'un langage moderne comme le C++ ne soit pas plus robuste ? Expliquez la ou les raisons derrière ces choix de design. **[2 points]**

2. Dans le code suivant, est-il vrai que `x` vaut `2.0` ? Justifiez votre réponse. **[2 points]**

```
1 int main() {
2     short tab[] = {0, 1, 2, 3, 4, 5, 6, 7};
3     double *ptr = (double *)(&tab[0]);
4     double x = ptr[2];
5     return 0;
6 }
```

Examen INFO-F105 Langages de Programmation 1; examen de juin 2021

Nom, prénom:
Numéro d'étudiant:

Numéro de question: 1

Les points importants dans la cotation ont été mis en gras entre parenthèses. La somme des points excède parfois le maximum car certaines parties de réponse, à elles seules, permettaient d'avoir le maximum. Il n'était cependant pas possible d'avoir plus de points que le maximum.

1. Ce code parcourt une liste (**1 point**) de nombres sur 16bits, stocke le maximum dans AX et le minimum dans BX, puis les affiche. (**3 points**) **[3 points]**

2. La différence entre le premier et le deuxième code est que dans le second code, on va chercher énormément d'opérandes dans la mémoire (RAM) et non dans les registres. (**1 point**)

   L'inconvénient principal de cette méthode est que les accès en RAM sont ($\approx 200\times$[1]) plus lents que les accès aux registres. (**2 points**)

   On peut aussi noter comme inconvénient que dans le second code, l'indice est recalculé à chaque fois que l'on accès à la mémoire, mais ce n'est pas le principal soucis. (**1 point**) **[3 points]**

[1] `https://gist.github.com/hellerbarde/2843375`

Numéro de question: 1

```
1  #include <iostream>
2
3  void fonction_mystere(const short valeurs[10]) {
4      short max = valeurs[0];
5      short min = max;
6      for(int i=9; i > 0; i--) {
7          if (valeurs[i] > max) {
8              max = valeurs[i];
9          } else if (valeurs[i] < min) {
10             min = valeurs[i];
11         }
12     }
13     std::cout << max << std::endl << min << std::endl;
14 }
15
16 int main() {
17     const short valeurs[10] =
18         {314, 399, 490, 98, 143, 379, 494, 437, 256, 435};
19     fonction_mystere(valeurs);
20     return 0;
21 }
```

La cotation partait de 4 si le code effectue le traitement requis (recherche de min/max), et diminuait pour chaque erreur. Les points ont été arrondis à l'unité inférieure. Entre autres:

- Oubli de l'affichage (`cout`) (**-1 point**)
- Utilisation de `const short max = ...` (**-1 point**)
- Des boucles imbriquées (**-2 points**)
- Utilisation de `int` à la place de `short` (**-0.5 points**)

N'ont **pas** fait perdre de points:

- Ajouter un `using namespace std;`
- Parcourir le tableau de 0 à 9 plutôt que de 9 à 0
- Initialiser `min` et `max` à `valeurs[0]` ou à une très grande/petite valeur
- Le nom des variables
- L'utilisation d'une variable intermédiaire pour stocker `valeurs[i]`

Examen INFO-F105 Langages de Programmation 1 ; examen de juin 2021

Nom, prénom:
Numéro d'étudiant:

Numéro de question: 2

---

1. La fonction `main` peut être placée avant les autres fonctions, à condition de les déclarer à l'aide de leur prototype soit dans le même fichier avant la fonction `main`, soit dans un header séparé qu'il faudra ensuite inclure dans le fichier actuel. **[1 point]**

2. Dans la `fonction1`, les paramètres sont passés **par valeur** : les valeurs des variables de la fonction `main` sont copiées dans des variables locales ; les modifications de ces variables par la `fonction1` n'auront pas d'impact sur les variables de `main`. Dans la `fonction2`, les paramètres sont passés **par référence** : leurs modifications dans la fonction vont également modifier les variables de `main`. Dans la fonction `fonction3`, les paramètres sont passés **par pointeur** : leurs modifications dans la fonction vont également modifier les variables de `main`. **[1 point]**

3. Une fonction a un effet de bord si elle modifie l'état du système en dehors de son environnement local. La `fonction1` n'a donc pas d'effet de bord, tandis que les fonctions `fonction2` et `fonction3` ont un effet de bord, vu qu'elles modifient directement les variables de `main`. **[1 point]**

4. Ces différents modes de passage des paramètres ont bien un effet sur la sécurité du programme, lié à leur effet de bord : la fonction appellante doit être consciente que ses variables peuvent potentiellement être modifiées. On pourrait aussi avoir un problème dans le cas où on passe plusieurs fois la même variable à une fonction qui utilise un passage par référence ou par pointeur. Le passage par valeur est donc plus sûr. **[1 point]**

5. Dans le cas de passage par valeur, les copies des paramètres peuvent être potentiellement coûteuses en temps et en mémoire pour les grosses données. Par contre, l'accès aux paramètres est moins efficace pour les fonctions utilisant le passage par référence ou par pointeur. **[1 point]**

6.
   - a=30, b=0, c=20
   - a=10, b=20, c=20
   - a=10, b=20, c=30

   [1 point]

7.
   - a=50, b=80, c=30
   - a=50, b=20, c=30

- a=50, b=20, c=30

[1 point]

8.
- a=0, b=20, c=30
- a=0, b=20, c=30
- a=0, b=20, c=30

[1 point]

9. Voir slide 42 du cours théorique sur les fonctions. **[1 point]**

10. Il s'agit de surcharge de fonction. Avec trois variables déclarées comme suit : `int a=10, b=20, c=30;`
on peut appeler la fonction `afficher_variables` de la ligne 4 comme suit : `afficher_variables(a,b,c);`
et la fonction `afficher_variables` de la ligne 8 comme suit : `afficher_variables(&a, &b, &c);` **[1 point]**

Examen INFO-F105 Langages de Programmation 1; examen de juin 2021

Nom, prénom:
Numéro d'étudiant:

Numéro de question: 3

```
1  #include <iostream>
2
3  unsigned fact(const unsigned &n) {
4     return (n > 1) ? n*fact(n-1) : 1;
5  }
6
7  class CoefB {
8  private :
9      unsigned _n;
10     unsigned _k;
11 public :
12     CoefB(const unsigned n=1, const unsigned k=1):_n(n), _k(k){}
13     ~CoefB() = default;
14     unsigned getN() const{ return _n; }
15     unsigned getK() const{ return _k;}
16     void setN(const unsigned &n) { _n = n;}
17     void setK(const unsigned &k) { _k = k;}
18     unsigned value() const{
19         return _n >= _k ? fact(_n)/
20                               (fact(_k)*fact(_n - _k)) : 0; }
21 };
22
23 std::ostream& operator<<(std::ostream &s, const CoefB &c) {
24     s << "(" << c.getN() << "," << c.getK() << ") = "
25                                                 << c.value();
26     return s;
27 }
28
29 int main(){
30     CoefB cb;
31     std::cout << cb << std::endl;
32     cb.setN(7); cb.setK(3);
33     std::cout << cb << std::endl;
34     CoefB cb2(5,3);
35     std::cout << cb2 << std::endl;
36     return 0;
37 }
```

Examen INFO-F105 Langages de Programmation 1; examen de juin 2021

Nom, prénom:
Numéro d'étudiant:

Numéro de question: 4

4.1.a publicTab[3]+5=13. La question met en oeuvre l'arithmétique de pointeurs; il n'est donc PAS nécessaire de connaître l'adresse de publicTab[2] pour répondre. Un certain nombre d'étudiants ont aussi oublié qu'on indexe les tableaux à partir de 0...

4.1.b Pas d'erreur à la compilation. Le dépassement de tableau ne viole pas les règles du langage, et sera donc accepté à la compilation. Voyez à ce sujet la section sur les tableaux en C/C++ dans le syllabus ou les transparents. (Notez que cette ligne peut potentiellement provoquer une erreur à l'exécution, mais cela n'est pas demandé par la question.)

4.1.c Pas d'erreur à la compilation. Cfr ci-dessus pour le dépassement de tableau; il y a de plus conversion implicite du float publicTab2[10] en int, qui est légale (cfr la section sur les conversions)

4.1.d Il y a bien sûr un dépassement de tableau. Les quatre premiers éléments imprimés proviennent de publicTab1; le dernier provient de l'emplacement mémoire juste après publicTab1[3]. A priori ici, toutes les variables locales de main sont stockées sur la pile de façon contigue, et on peut s'attendre à ce que la valeur 2 additionelle soit égale à secretTab[0].
(Pour votre information, sachez que certaines contre-mesures aux attaques "buffer overflow" consistent à ajouter de l'aléas sur la position de ces variables locales afin de les rendre moins prévisibles, et donc la vulnérabilité moins exploitable.)
Beaucoup d'étudiants ont répondu que le dernier 2 imprimé était publicTab1[0], et expliqué cela par une lecture cyclique du tableau. Cela n'est évidemment pas correct.

4.1.e Pas de dépassement possible des tableaux en Java. (cfr à nouveau la section sur les tableaux dans les transparents).

4.1.f Un des principaux choix de design du C++ est la rétro-compatibilité avec C, qu'on peut difficilement qualifier de langage moderne. Par ailleurs, le design de C privilégie l'efficacité sur la robustesse en offrant au programmeur un accès presque direct à la mémoire.

4.2 A priori non. La question synthétise votre compréhension des conversions de pointeurs, des conversions pointeur vers tableau et vice-versa, et de l'arithmétique de pointeurs.

ptr étant un pointeur vers double, la ligne 4 va chercher la variable contenue dans la zone mémoire de 8 octets (interprétée comme double) située 2*8 octets au-delà

de ptr[0]. Puisque tab est quant à lui un tableau de short (stockés chacun sur 2 octets), ptr[2] ne vaut certainement pas tab[2].

De façon générale, les points ont été attribués d'avantage sur base des justifications fournies que sur base des "réponses" (vrai/faux, oui/non, 13). Certaines réponses correctes mal justifiées ont donc pu amener une note réduite, alors que des réponses fausses mais démontrant une bonne compréhension de la matière ont valu beaucoup de points.