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 — juin 2023

- 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 interdits
- Durée de l'examen : deux heures trente
- 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

Considérons le code suivant écrit en NASM x86 (muni des macros SASM vues en séances d'exercices) :

```
1  section .data
2  ; tableaux de words (16 bits)
3  A dw 0, 1, 2, 3, 4, 4, 3, 2, 1, 0
4  B dw 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5  n dw 10
6
7  section .text
8  global CMAIN
9  CMAIN:
10     mov cx, 0
11     mov esi, A
12     mov edi, B
13 lab0:
14     mov ax, [esi]
15     mov bx, ax
16     xor bx, cx
17     test bx, 1
18     jz lab1
19     mov [edi], ax
20     mov word [esi], 0
21 lab1:
22     add esi, 2
23     add edi, 2
24     inc cx
25     cmp cx, [n]
26     jb lab0
27     xor eax, eax
28     ret
```

1. Décrivez brièvement ce que fait ce code (sans l'expliquer ligne par ligne). **[2 points]**
2. Que contiennent les tableaux `A` et `B` après l'exécution du programme ? **[2 points]**
3. L'instruction `jb` (ligne 26) peut-elle être emplacée par `jna` ? **[1 point]**
4. Le programme fonctionne-t-il toujours si la ligne 17 est remplacée par `and bx, 1` ? **[1 point]**
5. Peut-on utiliser `si` et `di` à la place de `esi` et `edi` ? Justifiez. **[1 point]**
6. Sur la page de réponse, complétez le code `C++` de manière à ce qu'il effectue le même traitement que le code assembleur. **[3 points]**

Question 2

Étant donné le code C++ de la page suivante, répondez aux questions suivantes :

1. Qu'affichera l'output à la fin de l'exécution du code ? Justifiez votre réponse. **[5 points]**

2. Indiquez le nom de la ou les fonction(s) où du masquage (*shadowing*) se produit, et justifiez votre réponse. **[1 point]**

3. Y aurait-t-il un problème si la ligne 38 ("`fonction3(tableau_2)`") était remplacée par "`fonction3(nullptr)`" ? Si oui, décrivez brièvement la nature du problème. **[2 points]**

4. Réécrivez le code de la `fonction4()` en remplaçant les références `a` et `b` par des pointeurs `a` et `b`. **[1 point]**

5. Le retour de la fonction `fonction4()` est problématique. Expliquez pourquoi. **[1 point]**

REMARQUES :

- En cas de valeur impossible à déterminer, veuillez indiquer une valeur possible et expliquer.
- Si vous identifiez une ou plusieurs erreurs, veuillez spécifier pour chaque erreur s'il s'agit d'une erreur syntaxique ou sémantique et expliquer.

Listing 1: Code de la Question 2

```
1  #include <iostream>
2
3  int x = 0;
4  void fonction1(int y) {
5     x = (x + y) % 2;
6  }
7  void fonction2(int* p, int indice, int valeur) {
8     int x = valeur + 2;
9     p[indice] = x;
10 }
11 int fonction3(int* p) {
12    static int i = 0, *p2 = nullptr;
13    if (p == nullptr)
14       ++i;
15    else
16       p2 = p;
17    return p2[i];
18 }
19 int* fonction4(int& a, int& b) {
20    a += b;
21    int c = 2 * a;
22    return &c;
23 }
24
25 int main() {
26    int y = 11, z = 0;
27    int tableau_1[] = { 10, 9, 8 };
28    int tableau_2[] = { 10, 9, 8 };
29
30    fonction1(y);
31    std::cout << "x = " << x << std::endl;
32    fonction2(tableau_1, 2, 4);
33    for (int i = 0; i < 3; ++i) {
34       z += tableau_1[i];
35    }
36    std::cout << "x = " << x << std::endl;
37    std::cout << "z = " << z << std::endl;
38    fonction3(tableau_2);
39    z = fonction3(nullptr);
40    std::cout << "z = " << z << std::endl;
41    z = 19;
42    fonction4(y, z);
43    std::cout << "y = " << y << std::endl;
44    return 0;
45 }
```

Question 3

Étant un bon développeur, en plus de pouvoir réparer les imprimantes et smartphones de vos amis, vous êtes également capable de programmer des logiciels rapidement, même lors d'une session d'examen. Votre ami, Mr. Reno Twinko, vous demande d'implémenter la base de son logiciel de gestion de vente. Voici ses instructions.

1. Implémentez une classe **Car** avec:

   - un ou plusieurs constructeurs. Les attributs de la classe sont les suivants: `brand`, `model`, `price` et `dealInfo`. La valeur par défault de `dealInfo` est la phrase "`no deal info`". **[1 point]**
   - un destructeur. **[1 point]**
   - les getters. **[1 point]**
   - les setters. **[1 point]**
   - une méthode `calculateDeal` prenant en paramètre un tableau de variables de type **Car**. Ce tableau peut contenir jusqu'à 5 éléments. Cette méthode change la valeur de l'attribut `dealInfo`. Si la moyenne des prix des voitures données en paramètre est plus élevée que le prix de l'objet actuel de la méthode, `dealInfo` contiendra "`This is a good deal !`". Si la moyenne est plus petite que le prix de l'objet actuel de la méthode, alors `dealInfo` contiendra "`BAD DEAL`". **[2 points]**

2. Surchagez les opérateurs > et <<

   - La surcharge de l'opérateur de comparaison >, de telle façon que `v1 > v2` retourne `true` si et seulement si le prix de la voiture `v1` est plus grande que le prix de la voiture `v2`. **[2 points]**
   - La surcharge de <<. Pour pouvoir faire:

     ```
     cout << car << endl;
     ```

     avec `car` une instance de `Car`. Par exemple,

     – Pour une voiture avec les attributs suivants `brand = "BMW"`, `model = "Serie 3"`, `price = 12999` et `dealInfo = "This is a good deal !"`, l'output affichera:

     ```
     BMW Serie 3 at 12999 euros:  This is a good deal !
     ```

     [2 points]

Question 4

Considérez le programme C++ suivant.

```
1  #include <iostream>
2  using namespace std;
3
4  int myfunction(int* x, int y) {
5      int a = *x;
6      int b = y;
7      int* c = x;
8      int* d = &a;
9      int* e = &y;
10     cout << "e = " << e << endl;
11     int& f = a;
12
13     int g = a / 4;
14     int* h = new int(g);
15     a = a *2 ;
16     cout << "*c=" << *c << endl << "*d=" << *d << endl;
17     cout << "f=" << f << endl << "*h=" << *h << endl;
18
19     struct {unsigned a; int b;} mystruct;
20     mystruct.a = 1 ; mystruct.b = -1;
21     union myunion {unsigned a; int b;}
22     myunion; myunion.a = 1; myunion.b = -1;
23     cout << "a=a? " << (mystruct.a == myunion.a) << endl;
24     cout << "b=b? " << (mystruct.b == myunion.b) << endl;
25
26     int * pi = new int[10]{3,1,4,1,5,9,2,6,5,3};
27     int i = (&(*(2 + &pi[2])))[-2] + 1;
28     cout << "i=" << i << endl;
29     long* pid = (long *) pi;
30     int j = (&(*(2 + &pid[2])))[-2] + 1;
31     cout << "j=" << j << endl;
32
33     delete h;
34     cout << "h=" << h << endl;
35
36     cout << pi[11] << endl;
37
38     return *x;
39 }
40
41 int main() {
42     int x = 10 ;
43     int y = 100 ;
```

```
44      cout<<"&y = "<< &y<<endl;
45      int z = myfunction(&x, y);
46      cout << "x = " << x << "  y = "
47           << y << "  z = " << z<< endl;
48      return 0;
49  }
```

Répondez aux questions suivantes en supposant que le programme est compilé de telle façon que **sizeof(long)=2*sizeof(int)**, et exécuté sur une architecture little-endian.

1. Les valeurs de `e` et `&y` affichées par les ligne 10 et 44 sont-elles identiques ? Justifiez votre réponse. **[1 point]**
2. Les lignes 13 et 14 produisent-elles des erreurs à la compilation et/ou lors de l'exécution depuis ces lignes ? Justifiez vos réponses. **[1 point]**
3. Qu'est-ce que le programme affichera aux lignes 16 et 17 ? Justifiez votre réponse. **[2 points]**
4. Qu'est-ce que le programme affichera aux lignes 23-24 ? Justifiez votre réponse. **[1 point]**
5. Qu'est-ce que le programme affichera à la ligne 28 ? Justifiez votre réponse. **[1 point]**
6. Qu'est-ce que le programme affichera à la ligne 31 ? Justifiez votre réponse. **[1 point]**
7. La ligne 34 produit-elle des erreurs à la compilation et/ou lors de l'exécution depuis cette ligne ? Justifiez vos réponses. **[1 point]**
8. La ligne 36 produit-elle des erreurs à la compilation et/ou à l'exécution ? Qu'en-est-il si on y remplace le 11 par 100000 ? Justifiez vos réponses. **[1 point]**
9. Qu'est-ce que le programme affichera en ligne 46 ? Justifiez votre réponse. **[1 point]**

SOLUTION Examen INFO-F105 Langages de Programmation 1 ; examen – juin 2023

NOM, PRENOM (en majuscules) :
Numéro d'étudiant :

Numéro de question : 1

---

1. Ce programme itère sur tous les éléments `A[i]` dans le tableau `A` et *déplace* ceux dont la parité est différente de celle de `i` dans le tableau `B` (en mettant l'entrée associée dans `A` à 0).

2. Le tableau `A` contient les valeurs suivantes : `0, 1, 2, 3, 4, 0, 0, 0, 0, 0`.
   Le talbeau `B` contient les valeurs suivantes : `0, 0, 0, 0, 0, 4, 3, 2, 1, 0`.

   En effet, les 5 premiers éléments du tableau `A` sont précisément égaux aux indices correspondant (ils ont donc en particulier la même parité) alors que les 5 suivants sont de parité opposée.

3. Non car l'instruction `jna` (jump if not above) correspond à `jbe` (jump if below or equal) qui n'est pas la même instruction que `jb` (jump if below). La différence est que `jna cx, [n]` ira une position trop loin dans les tableaux car fera un `n+1` ème tour de boucle.

4. Oui puisque `test` est très similaire à `and` : il fait exactement la même chose excepté l'écriture du résultat à la fin. L'instruction `and` met bien les flags à jour, le `jz` qui suit se comportera donc normalement. La seule différence ici est le fait que l'ancienne valeur de `bx` est perdue, ce qui n'est pas grave car ce n'est qu'une valeur temporaire.

5. `esi` et `edi` contiennent des adresses. Or nous sommes ici en mode 32 bits, les adresses font donc également 32 bits et ne peuvent être stockées dans les registres `si` et `di` qui sont des registres 16 bits.

6.

```
1  constexpr short n = 10;
2  short A[n] = {0, 1, 2, 3, 4, 4, 3, 2, 1, 0};
3  short B[n] = {};
4
5  int main() {
6      for(short i=0; i < n; ++i) {
7          if((A[i] ^ i) & 1) {
8              B[i] = A[i];
9              A[i] = 0 ;
10         }
11     }
12     return 0;
13 }
```

Ce code peut également s'écrire de manière encore plus similaire au code ASM proposé :

```
1  constexpr short n = 10;
2  short A[n] = {0, 1, 2, 3, 4, 4, 3, 2, 1, 0};
3  short B[n] = {};
4
5  int main() {
6      for(short i=0, *a=A, *b=B; i < n; ++i, ++a, ++b) {
7          short tmp = *a;
8          if((tmp ^ i) & 1) {
9              *b = tmp;
10             *a = 0 ;
11         }
12     }
13     return 0;
14 }
```

Examen INFO-F105 Langages de Programmation 1 ; examen – juin 2023

NOM, PRENOM (en majuscules) :
Numéro d'étudiant :

Numéro de question : 2

SOLUTION

1. Le programme affichera :

```
x = 1
x = 1
z = 25
z = 9
y = 30
```

2. La seule fonction avec du masquage est `fonction2()` car sa variable locale `x` masque la variable globale `x`.

3. Oui il y aurait un problème. En effet, au moment de retourner une valeur, l'évalutation de `p2[i]` reviendrait à exécuter `*((int*)nullptr + 1)` or ceci nous donnerait un résultat indéfini[1] et conduirait vraisemblalement à un crash dû à un accès non autorisé en mémoire (*segmentation fault*).

4. Réécriture de `fonction4()` :

```
1 int* fonction4(int* a, int* b) {
2     *a += *b;
3     int c = 2 * *a;
4     return &c;
5 }
```

5. La fonction `fonction4()` retourne l'adresse d'une variable locale dont la durée de vie se termine à la fin de la fonction. L'adresse `&c` référence donc désormais une zone mémoire invalide (`c` a été détruit) et y accéder provoque un comportement indéfini.

[1] Ajouter une valeur autre que 0 à un `nullptr` est un comportement indéfini (C++20 § 7.6.6, 4.3). Il n'était cependant pas attendu que vous parliez de comportements indéfinis mais il était attendu que vous indiquiez que tout accès en mémoire à partir d'un `nullptr` est à proscrire.

Examen INFO-F105 Langages de Programmation 1 ; examen – juin 2023

NOM, PRENOM (en majuscules) :
Numéro d'étudiant :

Numéro de question : 3

SOLUTION

```cpp
#include <iostream>

using namespace std;

// -0.5 if too many syntax errors / no coherence
// -2 if no class
class Car {
  // -1 if attrs not private
  // -1 if incorrect type
  // (need float or double for price, string-like for the rest),
  // -0.5 if type is numerical but not float/double for price
  private:
    string _brand, _model, _dealInfo;
    double _price;
  // -1 if methods not public
  public:
    // +1 if at least default and 1 other constr.
    // -0.5 if incorrect type
    // -0.5 if used copy but no const and ref.
    // valid default constr
    Car(string brand, string model, double price) {
      _brand = brand;
      _model = model;
      _price = price;
      _dealInfo = "no deal info";
    };
    // +1 if destr., default or {}
    ~Car() = default;
    // +1 if correct getters
    // -0.5 if no const
    // -0.5 if incorrect return type
    string getBrand() const {
      return _brand;
    }
    string getModel() const {
      return _model;
    }
```

```
38      string getDealInfo() const {
39        return _dealInfo;
40      }
41      double getPrice() const {
42        return _price;
43      }
44      // +1 if correct setters
45      // -0.5 if return type not void
46      // -0.5 if params type != attrs type
47      void setBrand(string brand) {
48        _brand = brand;
49      }
50      void setModel(string model) {
51        _model = model;
52      }
53      void setDealInfo(string dealInfo) {
54        _dealInfo = dealInfo;
55      }
56      void setPrice(double price) {
57        _price = price;
58      }
59      // +2 if calculateDeal
60      // -0.5 if incorrect return type
61      // -0.5 if incorrect param ref sig.
62      // -0.5 if const
63      // -0.5 if formula incorrect
64      void calculateDeal(const Car cars[5]) {
65        unsigned size = 5;
66        double average = 0;
67        for (int i=0; i < size; i++){
68          average += cars[i].getPrice();
69        }
70        average = average / size;
71        if (average > getPrice()) {
72          setDealInfo("This is a good deal !");
73        } else {
74          setDealInfo("BAD DEAL");
75        }
76      }
77      // +2 if >
78      // -0.5 if incorrect return type
79      // -0.5 if incorrect param ref sig.
80      // -0.5 if no const
81      // -0.5 if incorrect return value
82      bool operator>(const Car& car) const {
83        return getPrice() > car.getPrice();
```

```
84      }
85      // +2 if <<
86      // -0.5 if incorrect return type
87      //  or no friend if in class def.
88      // -0.5 if incorrect params
89      // -0.5 if incorrect stream output build
90      // -0.5 if no const
91      // if in class def, then need friend keyword
92  //    friend ostream& operator<<(ostream& os, const Car& car){
93  //      os << car.getBrand() << " " << car.getModel() << " at " << car
94  //      return os;
95  //    }
96  };
97
98  ostream& operator<<(ostream& os, const Car& car){
99    os << car.getBrand() << " " \
100   << car.getModel() << " at " \
101   << car.getPrice() << " euros: " \
102   << " " << car.getDealInfo();
103   return os;
104 }
105
106 int main() {
107   cout << "Q3" << endl;
108   Car car("BMW", "Serie 3", 12999.99);
109   Car cars[5] = {
110     Car("BMW", "Serie 1", 12999.99),
111     Car("BMW", "Serie 2", 22999.99),
112     Car("BMW", "Serie 3", 32999.99),
113     Car("BMW", "Serie 4", 42999.99),
114     Car("BMW", "Serie 5", 52999.99),
115   };
116   cout << car << endl;
117   car.calculateDeal(cars);
118   cout << car << endl;
119 }
```

Examen INFO-F105 Langages de Programmation 1 ; examen – juin 2023

NOM, PRENOM (en majuscules) :
Numéro d'étudiant :

Numéro de question : 4

SOLUTION

1. Non. Le pointeur e a comme valeur l'adresse de la variable locale y. Le passage des paramètres se faisant par copie, l'adresse de la variable locale y en ligne 9 est différente de celle de la variable y imprimée en ligne 44.

2. Aucune erreur. A la ligne 13, la division de deux entiers est implement définie comme l'arrondi inférieur de la fraction. A la ligne 14, on passe un entier en paramètre donc aucune raison d'avoir une erreur.

   Beaucoup d'étudiants ont mentionné une conversion de float vers int en ligne 13. Bien que la valeur de 10/4 est arrondie à 2, il n'y a pas ici de conversion au sens du langage, car par définition, le résultat de la division de deux int est un int. Il faudrait même introduire une conversion explicite pour avoir le résultat 2.5 !

3. $*c = 10$
   $*d = 20$
   $f = 20$
   $*h = 2$
   c pointe vers la même adresse que x ; celle-ci stocke une valeur 10 à l'entrée de la fonction et n'est pas modifiée ensuite
   d pointe vers le contenu de la variable a; celle-ci vaut 10 après la ligne 5 et 20 après la ligne 15
   f est une référence vers a, un alias, et vaut donc aussi 20 en ligne 17
   h pointe vers une zone de mémoire sur le tas initialisée à la valeur 2 (résultat de la division du int 10 par 4) en ligne 14, et pas modifiée ensuite

4. a=a? 0
   b=b? 1
   La variable mystruct est une concaténation de deux attributs mystruct.a et mystruct.b; la place qu'elle occupe en mémoire est deux fois la taille d'un int. En ligne 20, ces deux attributs sont assignés à 1 et -1. Les valeurs de mystruct.a et mystruct.b sont 1 et -1.
   L'union myunion est soit int soit unsigned; la place qu'elle occupe en mémoire est celle d'un seul int. En ligne 22, cet emplacement mémoire est d'abord assigné les bits de 1 (vu comme un unsigned) puis il lui est assigné les bits de -1 (vu comme un int). Les valeurs de myunion.a et myunion.b sont les valeurs correspondantes à ces bits respectivement, vus comme un unsigned ou un int, c'est-à-dire $2^32-1 = 4294967295$ et -1.
   On n'a donc pas d'égalité dans le premier cas, mais bien dans le deuxième cas.

Notez que le programme n'affichera jamais True ou False, mais bien 1 ou 0.

5. On a:
   $\&pi[2]$ est l'adresse de $pi[2]$, c'est-à-dire l'adresse de pi + 2* taille de int
   $(2 + \&pi[2])$ est l'adresse de $pi[4]$, c'est-à-dire l'adresse de pi + 4* taille de int
   $*(2 + \&pi[2]))$ est la valeur en $pi[4]$
   $(\&(*(2+\&pi[2])))$ est l'adresse de $pi[4]$, c'est-à-dire l'adresse de pi + 4* taille de int
   $(\&(*(2 + \&pi[2])))[-2]$ est la valeur en $pi[2]$, c'est-à-dire 4
   $(\&(*(2 + \&pi[2])))[-2] + 1 = pi[2] + 1 = 5$

6. En raisonnant comme pour la question précédente on a :
   $(\&(*(2+\&bpi[2])))[-2]$ est la valeur en $bpi[2]$, c'est-à-dire le long dont la représentation binaire correspond à $pi[4]||pi[5]$ (puisqu'on suppose qu'un long est deux fois plus long qu'un int)
   $(\&(*(2 + \&bpi[2])))[-2] + 1$ est le long dont la représentation binaire correspond à la concaténation de $pi[4] + 1$ et $pi[5]$
   la valeur de j est la conversion de cette valeur en int, c'est-à-dire son mot de poids faible, c'est-à-dire (vu qu'on est en little endian) $pi[4] + 1$, donc 6

7. Pas d'erreur à la compilation; la mémoire est désallouée mais le pointeur survit. Le pointeur devient un "pointeur fou" ; tenter d'accéder à la mémoire pointée pourrait mener à une erreur à l'exécution (tout dépend de l'implantation du delete) ; par contre, imprimer le pointeur lui-même comme ici (i.e. l'adresse stockée, pas la valeur de la zone de mémoire à cette adresse) n'amènera aucune erreur puisque cela ne nécessite aucun accès à la mémoire.

8. Pas d'erreur à la compilation. Un "buffer overflow" se produit à l'exécution. Probablement pas fatal avec 11 (on accèdera à la mémoire sur le tas immédiatement adjacente à pi), mais risque plus grand d'accéder à de la mémoire non allouée si on remplace 11 par 100000.

9. x = 10 y = 100 z = 10
   y est initialisé à 100 en ligne 43 et passé par copie à la fonction; il n'est donc jamais modifié
   x est initialisé à 10 en ligne 42; elle est passée en référence à la fonction qui pourrait donc a priori modifier sa valeur. Cependant, la ligne 5 ne fait que copier sa valeur; la ligne 7 copie son adresse dans c, mais c n'est plus modifié par la suite; aucune autre ligne du code de la fonction ne modifie x.
   Le raisonnement précédent montre que en ligne 38 la valeur de x est renvoyée.

De manière générale, j'ai souvent accordé plus de points à la justification qu'à la réponse elle-même.