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 2022

- 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 : 2 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

Contexte

Suite aux événements en Ukraine, le $MI6$ a mandaté l'agent 008 de connaître la technologie sur laquelle travaille la Russie. Craignant des agents doubles, Q et 008 ont convenu d'une méthode de chiffrement.
Après quelques jours sur le terrain, l'agent 008 a envoyé un message chiffré. Q a préparé le code de déchiffrement et, suivant la procédure, a imprimé le code avant de lancer l'exécution. Entretemps l'état russe, ayant eu vent de la fuite a déclenché une $E - Bomb$, une bombe électromagnétique, qui a détruit tout appareil electrique au sein du $MI6$.
Comme vous vous en doutez, Q est trop occupé à réinstaller le système et M vous demande de décrypter le message de 008.

Le code

```
1  %include "io.inc"
2  CPU 386
3  global CMAIN
4
5  section .data
6      ; message de 008
7      LIST DW 29797, 27492, 27488, 28006, 16707, 20289, 16706
8
9  section .text
10 CMAIN:
11     mov ecx, LIST
12     mov al, 0
13 bcle:
14     movzx edx, al
15     mov ax, [ecx + 2 * edx]
16     PRINT_CHAR ah
17     and al, 31
18     cmp al, 0
19     jnz bcle
20     xor eax, eax
21     ret
```

Pour vous simplifier la tâche, voici les nombres en binaire (codés sur 16 bits)

29797	0	1	1	1	0	1	0	0	0	1	1	0	0	1	0	1
27492	0	1	1	0	1	0	1	1	0	1	1	0	0	1	0	0
27488	0	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0
28006	0	1	1	0	1	1	0	1	0	1	1	0	0	1	1	0
16707	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1
20289	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0	1
16706	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0

L'instruction `PRINT_CHAR val` permet d'afficher le caractère associé au code ASCII `val`. Considérez le système ASCII suivant :

- `PRINT_CHAR 65` imprime sur l'output "A"
- `PRINT_CHAR 90` imprime sur l'output "Z"
- `PRINT_CHAR 97` imprime sur l'output "a"
- `PRINT_CHAR 122` imprime sur l'output "z"

On vous demande :

1. Expliquez ce que fait le code assembleur. **[4 points]**
2. Indiquez ce que le programme imprime sur l'output en respectant la casse (majuscules, minuscules). **[2 points]**
3. Si on remplace **DW** par **DD** (l. 7), que faudrait-il changer au code pour qu'il fonctionne et produise le même résultat ? **[2 points]**
4. Peut-on remplacer l'instruction `jnz bcle` par `jne bcle` à la ligne 19 ? Justifiez votre réponse. **[2 points]**

Question 2

Étant donné le code C++ donné à la page suivante, répondez aux questions ci-dessous et justifiez brièvement (2-3 lignes maximum) si demandé :

1. Indiquez quelle sera la valeur affichée lors de l'appel de la fonction `f1` à la ligne n° 38. **[1 point]**
2. Indiquez quelle sera la valeur affichée lors de l'appel de la fonction `f1` à la ligne n° 42. **[1 point]**
3. Qu'est-il affiché lors de l'appel de la fonction `f2` ? **[1 point]**
4. Comment réécrire la fonction `f3` sans référence ? Expliquez. **[1 point]**
5. Que feriez-vous si vous deviez déplacer la fonction `f3` dans un autre fichier ? **[1 point]**
6. Qu'affiche la ligne n° 26 ? **[1 point]**
7. Peut-on remplacer la ligne n° 40 par celle-ci : « `f3(&e);` » ? Justifiez. **[1 point]**
8. Dans la fonction `f4`, quelles sont les valeurs affichées pour `g` aux lignes n°31 et n° 33 ? **[1 point]**
9. Comment s'appelle le concept appliqué à la variable `g` dans le point précédent ? Existe-t-il également en Python ? **[1 point]**
10. Indiquez quelle sera la valeur affichée pour `g` à la ligne n° 43. **[1 point]**

```
1  #include <iostream>
2
3  void f1(int a, int b=3, int c=12) {
4      static int x = a+b+c;
5      std::cout << "somme dans f1 vaut: " << x << std::endl;
6  }
7
8  void f2(unsigned int d) {
9      switch(d) {
10     case 60:
11         std::cout << "M" << std::endl;
12     case 70:
13         std::cout << "L" << std::endl;
14     case 80:
15         std::cout << "O" << std::endl;
16         d = d+10; break;
17     case 90:
18         std::cout << "R" << std::endl;
19     default:
20         std::cout << "P" << std::endl;
21     }
22 }
23
24 void f3(int& e) {
25     for (int i = 4; i < 28; i += i-1) e++;
26     std::cout << "e dans f3 vaut : " << e << std::endl;
27 }
28
29 void f4(int g) {
30     for (int i = 4; i < 28; i += i-1) g++;
31     std::cout << "g dans f4 vaut: " << g << std::endl;
32     if (g > 0) int g = 13;
33     std::cout << "g dans f4 vaut: "  << g << std::endl;
34 }
35
36 int main() {
37     int e = 0, g = 0;
38     f1(0, 1);
39     f2(70);
40     f3(e);
41     f4(g);
42     f1(0, 0, 1);
43     std::cout << "e = " << e << ", g = " << g << std::endl;
44     return 0;
45 }
```

Question 3

Soient deux points A et B dans le plan cartésien, (x_A, y_A) les coordonnées du point A et (x_B, y_B) les coordonnées du point B, alors la distance entre A et B sur le plan se calcule comme suit :

$$|AB| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

1. Implémentez une classe **Point** avec :
   — un ou plusieurs constructeurs (par défaut $x = 0$ et $y = 0$) **[1 point]**
   — un destructeur **[1 point]**
   — les getters **[1 point]**
   — les setters **[1 point]**
   — une méthode `distance` qui prend un autre `Point` en paramètre et qui renvoie la distance AB, en utilisant la méthode ci-dessus. Pour calculer les puissances et racine carrée, vous pouvez utiliser les méthodes de la librairie `cmath` respectives : `std::pow(x, y)` et `std::sqrt(x)`. **[2 points]**
2. Surchagez les opérateurs + et << :
   — La surcharge de + retourne un `Point` équivalent à (x_A+x_B, y_A+y_B). **[2 points]**
   — La surcharge de << pour pouvoir faire :

   ```
   cout << point << endl;
   ```

   avec `point` une instance de `Point`. Par exemple, pour un point de coordonnées $x = 2$ et $y = 3$, l'output affichera :

   ```
   (2, 3)
   ```

   [2 points]

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

Question 4

Considérez le programme C++ suivant, où l'appel de fonction **strcpy(destination,source)** copie le tableau de caractères **source** vers le tableau de caractères **destination**. Cette fonction se comporte comme la fonction homonyme en C.

```
1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  unsigned myfunction(char* x, int y[], float z){
7      char a[] = "infoF105" ;
8      char* b = new char [9];
9      int* c = y;
10     unsigned int d = z;
11     int e = 19;
12     int* f = new int(e);
13     int* g = &(e);
14     *g -= 1;
15     double h = *g + 1 ;
16     cout << "e=" << e << "\n*f=" << *f <<"\n";
17     cout << "*g=" << *g << "\nh=" << h <<"\n";
18     int i = c[*(&c[3]-2)+1]-2;
19     //strcpy(a, x);
20     //strcpy(x, a);
21     y[1] = 5 ;
22     c[2] = 1;
23     return d;
24 }
25
26 int main() {
27     char alpha [] = "Theorie des Langages";
28     int beta [] = {2,3,5,7,11,13,17};
29     int gamma = -1;
30     unsigned res = myfunction(alpha,beta,gamma);
31     return 0;
32 }
```

1. La ligne 10 du programme produira-t-elle une erreur à la compilation et / ou à l'exécution ? Justifiez vos réponses. **[1 point]**
2. Après la ligne 13, est-ce que les valeurs de **f** et **g** sont égales ? Justifiez votre réponse. **[1 point]**
3. Que se passera-t-il si on remplace **e** par **e+1** dans les lignes 12 et 13 ? Justifiez votre réponse. **[1 point]**
4. Qu'est-ce que le programme imprimera en lignes 16 et 17 ? Justifiez vos réponses. **[1 point]**
5. Quelle est la valeur de **i** après la ligne 18 ? Justifiez votre réponse. **[1 point]**
6. Si on décommente les lignes 19 et 20, produiront-elles une erreur à la compilation et / ou à l'exécution ? Qu'en est-il si on remplace aussi **a** par **b** dans les lignes 19 et 20 ? Justifiez vos réponses. **[1 point]**
7. Quelles seront les valeurs du tableau **beta** après la ligne 30 ? Justifiez votre réponse. **[1 point]**
8. Quelle est la valeur de **res** après la ligne 30 ? Justifiez votre réponse. **[1 point]**
9. Le programme viole un certain nombre de bonnes pratiques en C++. Donnez un exemple et suggérez une amélioration. **[2 points]**

Examen INFO-F105 Langages de Programmation 1 ; examen de juin 2022

Nom, prénom :
Numéro d'étudiant :

Numéro de question : 1

---

1. Le code permet de parcourir une liste chaînée d'éléments de 16 bits. `AH`, les 8 bits de poids fort de `AX`, correspond au caractère à imprimer sur l'output. Pour `AL`, les 8 bits de poids faible de `AX`, on applique un masque pour ne garder que les 5 bits de faible et le résutat obtenu correspond à l'indice de l'élément suivant à traiter. Le premier élément à traiter est le 0 et le parcours de la liste s'arrête quand l'indice de l'élément suivant est 0.
2. `tOkAmAk`, un tokamak est un dispositif de confinement magnétique expérimental explorant la physique des plasmas et les possibilités de produire de l'énergie par fusion nucléaire (source Wikipedia).
3. **DW** correspond à des mots de 16 bits.
   DD correspond à des mots de 32 bits.
   Chaque élément de la liste est donc 2 x plus long, avec les 16 bits de poids fort qui valent 0. Il convient donc de modifier la ligne 15 et de remplacer `[ecx + 2 * edx]` par `[ecx + 4 * edx]`
4. Oui, on peut remplacer l'instruction `jnz bcle` par `jne bcle` à la ligne 19. D'un point de vue conceptuel, les 2 instructions vérifie la valeur du `ZeroFlag` et d'un point vue technique, l'opcode des 2 instructions est le même. Il s'agit donc bien de la même opération pour le processseur.

Examen INFO-F105 Langages de Programmation 1 ; examen de juin 2022

Nom, prénom :
Numéro d'étudiant :

Numéro de question : 2

---

1. 13.
2. 13.
3. `"L\nO\n"` car il n'y a pas de `break` entre `case 70` et `case 80`.
4. On peut réécrire la fonction `f3` en passant un pointeur en paramètre. En effet, tout comme une référence, un pointeur passé en paramètre permet de modifier la variable vers laquelle il pointe (i.e. pas une copie).
5. Il faut déclarer le prototype de la fonction dans le code et avant la fonction `main`, par exemple dans un header séparé qu'il faudra veiller à inclure dans le fichier actuel.
6. `e dans f3 vaut : 4`.
7. Non, cela déclancherait une erreur à la compilation (`invalid conversion from 'int*' to 'int'`). En effet, la fonction attend un entier et reçoit un pointeur, or en C++ une conversion de pointeur vers int est illégale.
8. 4 et 4.
9. Le shadowing/masquage. Non, il n'existe pas en Python.
10. 0.

Remarques sur les corrections :

1. On obtient `a=0`, `b=1` et `c=12` donc la fonction `f1` affiche "13" (0+1+12).
2. Le mot-clef `static` désigne, sur une variable locale à un bloc, que la variable a la durée de vie d'une variable globale (donc une unique initialisation) mais a une portée limitée à son bloc de définition (et ses sous-blocs). La fonction affiche donc à nouveau 13 étant donné que la variable static int `x` a été initialisée lors de l'appel de la ligne 38.
3. En effet, la variable `d` est évaluée une seule fois et est ensuite comparée aux différents blocs `case`. Le code correspondant au bloc `case 70` est donc exécuté. Comme aucun `break` n'est présent dans ce `case:` à la fin de l'exécution du bloc, le `case` suivant est exécuté. Le bloc `case 90` n'est pas exécuté car la variable `d` n'est pas réévaluée. Default est seulement executé si aucun `case` n'est atteint, ce qui n'est pas le cas ici.
4. /
5. /
6. Les réponses indiquant uniquement 4 on été également acceptées. `i` prend les valeurs suivantes : 4, 7, 13 et 25, on rentre donc 4 fois dans la boucle. La variable `e` etant initialisée à 0 et incrémentée de 1 à chaque passage dans la boucle, elle vaut donc 4 à la ligne n°26. Les réponses telles que `e+4` ont également été acceptées.

7. /
8. En effet, la variable `g` qui est initialisée à la ligne n°32 n'existe que dans son scope. La variable `g` de la ligne 31 et 33 sont donc une seule et même variable qui est distincte de celle de la ligne 32.
9. Le shadowing est la déclaration d'une nouvelle variable dans un sous-bloc ayant le même nom qu'une variable déclarée dans un bloc parent sans ambiguïté.
10. Pour la fonction `f4`, il s'agit d'un passage par valeur (une copie de `g` est utilisée localement) et donc la valeur de `g` n'est pas modifiée dans le main, et reste à 0.

Examen INFO-F105 Langages de Programmation 1 ; examen de juin 2022

Nom, prénom :
Numéro d'étudiant :

Numéro de question : 3

```
1
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  // -0.5 if too many syntax errors / no coherence
8  // -2 if no class
9  class Point {
10   // -1 if attrs not private
11   // -1 if incorrect type (need float or double),
12   // -0.5 if type is numerical but not float/double
13   private:
14     double _x, _y;
15   // -1 if methods not public
16   public:
17     // +1 if at least default and 1 other constr.
18     // -0.5 if incorrect type
19     // -0.5 if used copy but no const and ref.
20     // valid default constr
21     Point(double x = 0, double y = 0) {_x = x; _y = y;};
22     Point(const Point& p): _x(p.getX()), _y(p.getY()) {}
23     // +1 if destr., default or {}
24     ~Point() = default;
25     // +1 if correct getters
26     // -0.5 if no const
27     // -0.5 if incorrect return type
28     double getX() const {
29       return _x;
30     }
31     double getY() const {
32       return _y;
33     }
34     // +1 if correct setters
35     // -0.5 if return type not void
36     // -0.5 if params type != attrs type
37     void setX(double x) {
38       _x = x;
```

```
39      }
40      void setY(double y) {
41        _y = y;
42      }
43      // +2 if distance
44      // -0.5 if incorrect return type
45      // -0.5 if incorrect param ref sig.
46      // -0.5 if not const
47      // -0.5 if formula incorrect:
48      //  no sqrt, no pow or no other point
49      double distance(const Point& p) const {
50        return sqrt(pow(p.getX() - getX(), 2)
51                    + pow(p.getY() - getY(), 2));
52      }
53      // +2 if +
54      // -0.5 if incorrect return type
55      // -0.5 if incorrect param ref sig.
56      // -0.5 if no const
57      // -0.5 if incorrect return value
58      Point operator+(const Point& p) const {
59        return Point(p.getX() + getX(), p.getY() + getY());
60      }
61      // +2 if <<
62      // -0.5 if incorrect return type
63      //  or no friend if in class def.
64      // -0.5 if incorrect params
65      // -0.5 if incorrect stream output build
66      // -0.5 if no const
67      // if in class def, then need friend keyword
68  //    friend ostream& operator<<(ostream& os, const Point& p){
69  //      os << "(" << p.getX() << ", " << p.getY() << ")";
70  //      return os;
71  //    }
72  };
73
74  ostream& operator<<(ostream& os, const Point& p){
75    os << "(" << p.getX() << ", " << p.getY() << ")";
76    return os;
77  }
78
79
80  // This was also accepted for operator+
81  // Point operator+(Point p1, const Point& p2){
82  //   return Point(p1.getX() + p2.getX(), p1.getY() + p2.getY());
83  // }
84
```

```
85  int main() {
86    cout << "Q3" << endl;
87    Point defaultp, point(2, 3), other(6, 7);
88    cout << "default constr.: " << defaultp << endl;
89    cout << "Get x: " << point.getX() << endl;
90    cout << "Get y: " << point.getY() << endl;
91    cout << "Set x = 4" << endl;
92    point.setX(4);
93    cout << "set y = 5" << endl;
94    point.setY(5);
95    cout << "Get x: " << point.getX() << endl;
96    cout << "Get y: " << point.getY() << endl;
97    cout << "Distance: " << point.distance(other) << endl;
98    Point result = point + other;
99    cout << "+: " << (point + other) << endl;
100 }
```

Examen INFO-F105 Langages de Programmation 1 ; examen de juin 2022

Nom, prénom :
Numéro d'étudiant :

Numéro de question : 4

---

1. En ligne 10, la valeur **z** de type **float**, est convertie automatiquement en **unsigned int** : pas d'erreur ni à la compilation ni à l'exécution.

   Notez que la conversion de valeur -1 résulte en un dépassement d'entier (la valeur est typiquement $2^{32} - 1$ soit 4294967295) ; cependant ceci ne génère pas d'erreur à l'exécution (c'est le comportement défini par le langage, et peut/doit a priori être l'intention du programmeur).

   Selon les paramètres passés au compilateur, celui-ci pourrait générer un avertissement lié à la conversion. (Mais pas d'erreur.)

2. La ligne 12 alloue, crée et initialise une variable entière sur le tas, de valeur égale à 19, et déclare et initialise un pointeur **f** dont la valeur est l'adresse de la variable créée sur le tas. La ligne 13 crée un pointeur dont la valeur est l'adresse de **e** ; les deux valeurs sont donc différentes.

3. Le changement en ligne 12 ne pose pas d'erreur à la compilation. Si seul ce changement est effectué, la valeur de la variable créée sur le tas vaudra simplement 20 au lieu de 19. Par contre, le changement en ligne 13 amènera une erreur à la compilation, car une variable temporaire n'a pas d'adresse.

   Notez que sans les parenthèses en ligne 13, il n'y aurait pas d'erreur à la compilation : la variable g vaudrait alors l'adresse de e, incrémentée de la taille d'un entier.

4. Le programme imprime
   e=18
   *f=19
   *g=18
   h=19
   Après la ligne 13, **g** est un alias de **e**, et donc la modification en ligne 14 affecte la valeur de **e**.

5. Valeur est $c[c[1] + 1] - 2 = c[4] - 2 = 11 - 2 = 9$

6. Pas d'erreur à la compilation mais un risque imporant d'erreur à l'exécution : en effet la fonction **strcpy** ne vérifie pas la taille des entrées et causera dans ce cas-ci un *buffer overflow*. Comme celui-ci se produira sur le tas, les chances sont grandes d'amener des bugs dans le code, des problèmes de sécurité, voire un crash du programme. (Certains compilateurs modernes ajoutent automatiquement du code pour détecter l'overflow et arrêter le programme.) Dans ce cas-ci, il est probable que les éléments immédiatement adjacents au tableau seront altérés ; mais potentiellement les paramètres, valeurs de retour et adresse de retour de la fonction peuvent aussi être affectées.

Si on remplace **a** par **b**, également *buffer overflow* pour les mêmes raisons. L'effet d'un tel bug accidentel peut être moins important sur le tas que sur la pile car les données du tas n'affectent pas directement le flux d'exécution d'un programme. (Néanmoins, des attaques *buffer overflow* sur le tas sont également possibles.)

7. Un tableau étant passé comme un pointeur, les valeurs de celui-ci sont en fait passées par référence. De même, le pointeur **c** pointe vers la même zone de mémoire. Les modifications en lignes 22 et 23 à l'intérieur de la fonction affecteront donc le tableau dans l'environnement appelant, qui aura alors comme valeurs 2,5,1,7,11,13,17
8. En ligne 10, la valeur **z** de type **float**, est convertie automatiquement en **unsigned int** : pas d'erreur ni à la compilation ni à l'exécution. La conversion de valeur -1 résulte cependant en un dépassement d'entier : la valeur est $2^{sizeof(int)} - 1$ soit 4294967295 pour des *int* de 32 bits
9. Une réponse évidente est l'absence de **delete** (et réinitialisation du pointeur) suivant un **new**. Mais on peut aussi citer l'absence de commentaires et d'aération du code, des noms peu explicites, des variables non utilisées, des conversions à tout va. . .