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 d'août 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 : 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)

Solutions included ! ! ! ! !

Question 1

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

```
1  SECTION .data
2  a dw 10, 9, 6, 7, 8, 3, 1
3  n dw 7
4
5  section .text
6  global CMAIN
7  CMAIN:
8      movzx ecx, word [n]
9      dec   ecx
10 iter:
11     mov   edx, ecx
12     dec   edx
13     shr   edx, 1
14     mov   bx, [a + ecx*2]
15     cmp   bx, [a + edx*2]
16     jg    no
17     dec   ecx
18     cmp   ecx, 1
19     jne   iter
20     mov   eax, 1
21     jmp   over
22 no: mov   eax, 0
23 over:
24     PRINT_UDEC 4, eax
25     ret
```

1. Décrivez en une phrase ce que fait ce code. **[2 points]**
2. Qu'affiche le PRINT_UDEC en ligne 24 ? **[1 point]**
3. Peut-on utiliser uniquement cx (et pas ecx) ? Justifiez. **[1 point]**
4. Pourquoi le registre bx est-il utilisé aux lignes 14-15 au lieu de ebx ? **[1 point]**
5. Exemplifiez l'utilité d'un saut non conditionnel en expliquant à quoi sert la ligne 21. **[2 points]**
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 ? **[5 points]**
2. Réécrivez `funct2` sans la boucle `for` et avec une boucle `while` en optimisant ou simplifiant la fonction si possible. Le `cout` n'est pas nécessaire. **[1 point]**
3. Réécrivez `funct3` sans la boucle `while` et avec une boucle `for` en optimisant ou simplifiant la fonction si possible. Le `cout` n'est pas nécessaire. **[1 point]**
4. Qu'affichera l'output de la ligne 38 si on remplace `z` par `*z` ? Justifiez votre réponse. **[1 point]**
5. Expliquez le *shadowing* et donnez un exemple. **[2 points]**

REMARQUES :

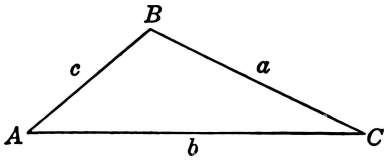
— En cas de valeur impossible à déterminer, veuillez indiquer une valeur possible et expliquer.

— Si vous identifiez une ou plusieurs erreurs, veuillez spécifier pour chaque erreur s'il s'agit d'une erreur syntaxique ou sémantique et expliquer.

```
1  #include <iostream>
2
3  void funct1(int a = 2) {
4    static int x = 1;
5    x += a;
6    std::cout << "funct1 = " << x << std::endl;
7    x++;
8  }
9
10 void funct2(int& x) {
11   for (int i = x - 1; i > 0; i--) {
12     x *= i;
13   }
14   std::cout << "funct2 = " << x << std::endl;
15 }
16
17 void funct3(int* x) {
18   int i = 2 * (*x - 1);
19   while (i - 1 > -1) {
20     *x += i / 2;
21     i -= 2;
22   }
23   std::cout << "funct3 = " << * x << std::endl;
24 }
25
26 int main() {
27   int x = 4, y = 5;
28   int* z = &y;
29   funct1(1);
30   std::cout << "x = " << x << std::endl;
31   funct2(x);
32   funct3(z);
33   funct1(3);
34   std::cout << "x = " << x << std::endl;
35   funct1();
36   std::cout << "x = " << x << std::endl;
37   std::cout << "y = " << y << std::endl;
38   std::cout << "z = " << z << std::endl;
39   return 0;
40 }
```

Question 3

En géométrie euclidienne, un triangle est une figure plane formée par trois points (appelés sommets) et par les trois segments qui les relient (appelés côtés). Dans l'exercice suivant, nous représenterons un triangle par les longueurs des trois côtés.



1. Implémentez une classe **Triangle** avec :
   - un ou plusieurs constructeurs (par défaut $(a = 1, b = 1, c = 1)$). **[1 point]**
   - un destructeur. **[1 point]**
   - les getters. **[1 point]**
   - les setters. **[1 point]**
   - une méthode `surface` qui calcule l'aire du triangle. Pour rappel d'après la formule d'Héron l'aire d'un triangle est égale à $\sqrt{p \times (p-a) \times (p-b) \times (p-c)}$ où $p = \frac{1}{2} \times (a + b + c)$. Pour calculer la racine carée, vous pouvez utiliser la méthode de la librairire `cmath` : `std::sqrt(x)`. **[1 point]**
   - une méthode `estValide` permettant de determiner si le triangle respecte l'inégalité triangulaire qui stipule que la longueur d'un côté est inférieure à la somme des longueurs des deux autres côtés. **[1 point]**
2. Surchagez les opérateurs > et <<
   - La surcharge de l'opérateur de comparaison >, de telle façon que s>t retourne `true` si et seulement si l'aire du triangle s est plus grande que l'aire du triangle t. N'hésitez pas à faire appel à la méthode `surface` définie précédemment. **[2 points]**
   - La surcharge de <<. Pour pouvoir faire :

     ```
     cout << triangle << endl;
     ```

     avec `triangle` une instance de `Triangle`. Par exemple,
     - Pour un triangle avec les attributs suivants $a = 2$, $b = 5$ et $c = 6$, l'output affichera :

       ```
       Le triangle avec les caractéristiques suivants
       a = 2, b = 5 et c = 6 est un triangle valide car il
       respecte l'inégalité triangulaire.
       ```
     - Pour un triangle avec les attributs suivants $a = 1$, $b = 2$ et $c = 3$, l'output affichera :

       ```
       Le triangle avec les caractéristiques suivants
       a = 1, b = 2 et c = 3 n'est pas un triangle valide
       car il ne respecte pas l'inégalité triangulaire.
       ```
       [2 points]

Question 4

Considérez le programme C++ suivant, et répondez aux questions en page suivante. On suppose une architecture IA32.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int myfunction(int a, int b) {
6      a = a;
7      int c = a + b ;
8      int d = a - b ;
9      int e = &c - &d ;
10     cout<<"e="<< e<<endl;
11
12     for(int i = 0 ; i < 7; i++) {
13         cout<<"valeur suivante = "<< (&e)[i]<<endl;
14     }
15
16     //cout<<"valeur suivante = "<< (&e)[10000]<<endl;
17     //cout<<"i = "<< i<<endl;
18
19      int* f = &(a);
20      int* g = new int(*f) ;
21      a = 2*a ;
22      cout<<"*f = "<< *f<< "  *g="<< *g<<endl;
23
24      char h [] = {1,3,5,7,9,11,13,15,17,19};
25      int m = h[((&h[1])+2)[-1]];
26      cout<<"m = "<< m<<endl;
27      int n = ((int*) h)[0];
28
29      int *o = g;
30      delete g;
31      //cout<<"*o = "<< *o<<endl;
32
33     return a;
34 }
35
36 int main() {
37     int x = 100 ;
38     double y = 1000 ;
39     myfunction(x,y);
40     cout<<"x = "<< x<<"  y = "<< y<<endl;
41     return 0;
42 }
```

1. Qu'est-ce que le programme affichera en ligne 10 ? Justifiez votre réponse. **[1 point]**
2. Qu'est-ce que le programme affichera pour les lignes 12-14 ? Justifiez votre réponse. **[2 points]**
3. Si on décommente la ligne 16, le programme produira-t-il une erreur à la compilation et/ou à l'exécution ? Justifiez vos réponses. **[1 point]**
4. Si on décommente la ligne 17, le programme produira-t-il une erreur à la compilation et/ou à l'exécution ? Justifiez vos réponses. **[1 point]**
5. Qu'est-ce que le programme affichera en ligne 22 ? Justifiez votre réponse. **[1 point]**
6. Qu'est-ce que le programme affichera en ligne 26 ? Justifiez votre réponse. **[1 point]**
7. Après exécution de la ligne 27, la variable n vaut-elle 0, 1, 3, 5, -1 ou une autre valeur ? Justifiez votre réponse. **[1 point]**
8. Si on décommente la ligne 31, le programme produira-t-il une erreur à la compilation et/ou à l'exécution ? Justifiez vos réponses. **[1 point]**
9. Qu'est-ce que le programme affichera en ligne 40 ? Justifiez votre réponse. **[1 point]**

Examen INFO-F105 Langages de Programmation 1 ; examen d'août 2022

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

Numéro de question : 1

---

1. Ce programme effectue la vérification que le tableau `a` (stocké comme variable globale) satisfait la relation d'ordre de (max-)heap : l'élément à l'indice `i` a ses enfants (s'ils existent) aux indices `2*i+1` et `2*i+2` donc le parent de l'élément à l'indice `i` est à la position `(i-1) / 2` (si `i > 0`). Le programme parcourt le tableau de droite à gauche et vérifie si l'élément à la positions `i` est strictement plus grand que son parent et affiche 0 si c'est le cas. Sinon, une fois tout le tableau parcouru, le programme affichera 1.
2. Puisque le tableau fourni satisfait la relation d'ordre de heap, le programme affichera 1 : la condition de la ligne 16 ne sera jamais vérifiée et donc le `mov eax, 0` de la ligne 22 ne sera jamais exécuté.
3. `cx` peut tout à fait servir de compteur ici puisque le tableau est loin de nécessiter 32 bits d'indiçage. Cependant l'adressage tel que présenté dans les lignes 14 et 15 impose l'utilisation d'un registre 32 bits dans sa syntaxe, i.e. `cx` ne serait pas accepté.
4. Le tableau `a` contient des entiers codés sur 16 bits (ce qui est exprimé par le `dw` à la ligne 2) et `bx` fait 16 bits, donc parfait pour stocker temporairement ces valeurs. Elles peuvent également être mises dans `ebx` à condition d'étendre la représentation (avec `movzx`) mais `cmp` infère la taille de l'opérande mémoire sur base de la taille de l'opérande registre, ce qui causerait la lecture de deux nombres à la fois (i.e. un bug).
5. Un saut inconditionnel sert – par exemple – à implémenter la clause `else` d'un saut conditionnel : si une paire d'indices `(i, j)` violant la relation d'ordre de heap est trouvée, le programme ira directement à la ligne 22 pour mettre `eax` à 0. Il faut cependant faire attention à ne pas exécuter cette instruction si le programme se déroule bien et que la condition (l. 15-16) n'est jamais vérifiée. Pour cela, il faut ajouter un saut inconditionnel à la sortie de la boucle, permettant d'aller directement après ce traitement conditionnel.

6.

```
#include <iostream>

short a[] = {10, 9, 6, 7, 8, 3, 1};

int main() {
    bool loop = true;
    int i = 7;
    while(i > 0 and loop) {
        --i;
        int parent = (i-1) / 2;
        if(a[i] > a[parent])
            loop = false;
    }
    std::cout << (loop ? 1 : 0) << std::endl;
    return 0;
}
```

Examen INFO-F105 Langages de Programmation 1 ; examen d'août 2022

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

Numéro de question : 2

---

1. l'ouput affichera :

```
funct1 = 2
x = 4
funct2 = 24
funct3 = 15
funct1 = 6
x = 24
funct1 = 9
x = 24
y = 15
z = 0x7fffa89036fc // valeur possible, adresse mémoire de y
```

2. Réécriture de `funct2`

```
1 void funct2(int& x){
2     int i = x - 1;
3     while ( i > 0){
4         x *= i;
5         i--;
6     }
7 }
```

3. Réécriture de `funct3`

```
1 void funct3(int* x){
2     for( int i = *x - 1; i > 0; i--){
3         *x += i;
4     }
5 }
```

4. Remplacement `z` par `*z`, l'output affichera :

```
*z = 15
```

5. Le *shadowing*, en programmation, est une technique qui consiste à redéclarer dans un scope une variable déjà déclarée avant le scope considéré. Toutes les opérations suivantes dans le scope ayant la variable comme opérante se réfèreront à la variable redéclarée. Quand on sortira du scope, on oubliera la variable redéclarée et on ne considèrera que la variable initiale.

```
1 #include <iostream>
```

```
2
3  int main(){
4      int i = 5;
5      {
6          std::cout << i << std::endl;
7          int i= 0;
8          std::cout << i << std::endl;
9          i = 10;
10         std::cout << i << std::endl;
11     }
12     std::cout << i << std::endl;
13     return 0;
14 }
```

affichera

```
5
0
10
5
```

Examen INFO-F105 Langages de Programmation 1 ; examen d'août 2022

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

Numéro de question : 3

```
1
2
3
4  #include <iostream>
5  #include <cmath>
6
7  using namespace std;
8
9  // -0.5 if too many syntax errors / no coherence
10 // -2 if no class
11 class Triangle
12 {
13   // -1 if attrs not private
14   // -1 if incorrect type (need float or double),
15   // -0.5 if type is numerical but not float/double
16   private:
17
18   double _a, _b, _c; // assign the variables for the sides
19
20   // -1 if methods not public
21   public:
22
23     // +1 if at least default and 1 other constr.
24     // -0.5 if incorrect type
25     // -0.5 if used copy but no const and ref.
26     // valid default constr
27     Triangle();
28     Triangle(const double a = 1, const double b = 1,
29             const double c = 1);
30     Triangle(const Triangle& t): _a(t.getA()), _b(t.getB()),
31     _c(t.getC()) {}
32
33     // +1 if destr., default or {}
34     ~Triangle() = default;
35
36
37
38
```

```
39      // +1 if correct getters
40      // -0.5 if no const
41      // -0.5 if incorrect return type
42      double getA() const {
43        return _a;
44      }
45      double getB() const {
46        return _b;
47      }
48      double getC() const {
49        return _c;
50      }
51
52      // +1 if correct setters
53      // -0.5 if return type not void
54      // -0.5 if params type != attrs type
55      void setA(double a) {
56        _a = a;
57      }
58      void setB(double b) {
59        _b = b;
60      }
61      void setC(double c) {
62        _c = c;
63      }
64
65      // +1 if estValide
66      // -0.5 if incorrect return type
67      // -0.5 if incorrect param ref sig.
68      // -0.5 if not const
69      // -0.5 if formula incorrect:
70      // -0.5 if getters not used
71      // -0.5 if each side ==0 is not checked
72      bool estValide() const{
73        return ((getA() < getB() + getC()) &&
74        (getB() < getA() + getC()) &&
75        (getC() < getA() + getB())
76        || (getA() == 0) || (getB() == 0) || (getC() == 0));
77      }
78
79
80
81
82
83
84
```

```
85      // +1 if surface
86      // -0.5 if incorrect return type
87      // -0.5 if incorrect param ref sig.
88      // -0.5 if not const
89      // -0.5 if formula incorrect:
90      // - 0.5 if getters not used
91      double surface() const {
92        p = 0.5 * (getA() + getB() + getC())
93        return sqrt( p * (p-a) * (p-b) * (p-c));
94      }
95
96      // +2 if >
97      // -0.5 if incorrect return type
98      // -0.5 if incorrect param ref sig.
99      // -0.5 if no const
100     // -0.5 if incorrect return value
101     bool operator >(const Triangle& t1,
102                     const Triangle& t2 ) const {
103       return t1.surface() > t2.surface();
104     }
105
106     // +2 if <<
107     // -0.5 if incorrect return type
108     //  or no friend if in class def.
109     // -0.5 if incorrect params
110     // -0.5 if incorrect stream output build
111     // -0.5/1 if incorrect text
112         (copied/pasted from another class)
113     // -0.5 if no const
114     // if in class def, then need friend keyword
115
116 //    friend ostream& operator<<(ostream& os,
117                                  const Triangle& t){
118 //     ...
119 //      return os;
120 //    }
121 };
122
123 ostream& operator<<(ostream& os, const Triangle& t){
124   if (t.estValide()){
125     os << "Le triangle avec les caracteristiques suivants"
126     <<"a = "<<t.getA()<< ", b = " << t.getB() << ", "
127     <<"c = "<<t.getC()
128     <<"est un triangle valide car il respecte
129         l'inegalite triangulaire.";
130   }
```

```
131   else {
132     os << "Le triangle avec les caracteristiques suivantes"
133     <<"a = "<<t.getA()<< ", b = " << t.getB() << ", "
134     <<"c = "<<t.getC() <<"n'est pas un triangle valide
135     car il ne respecte l'inegalite triangulaire.";
136   }
137   return os;
138 }
```

Examen INFO-F105 Langages de Programmation 1 ; examen d'août 2022

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

Numéro de question : 4

---

1. La valeur de e est la différence entre les adresses de c et d. Cette valeur n'est à strictement parler pas définie par le langage C++. Pour l'architecture donnée, les variables c et d seront a priori stockées de façon consécutive sur la pile, et la différence vaut donc -1.
2. Le programme affiche le contenu en mémoire suivant la variable e, plus exactement la mémoire correspondant à 7 variables entières. (Notez ici la conversion automatique de l'adresse d'une variable de type int en un tableau de int.)

   Pour répondre à cette question de façon plus précise, il faut utiliser la représentation des données en mémoire sur la pile.

   La fenêtre correspondant à la fonction *myfunction* contiendra les paramètres d'appel, suivis de l'adresse de retour (une adresse dans le segment TEXT), suivie du contenu du stack pointeur (une adresse dans la pile), suivi des paramètres locaux dans l'ordre d'apparition.

   Un exemple d'affichage est alors le suivant :

```
1 valeur suivante = -1
2 valeur suivante = -900
3 valeur suivante = 1100
4 valeur suivante = 32764
5 valeur suivante = 1708086216
6 valeur suivante = 100
7 valeur suivante = 1000
```

3. Pas d'erreur à la compilation. Erreur probable à l'exécution (segmentation fault) car on va tenter d'accéder à de la mémoire non allouée ou protégée par le système d'exploitation
4. Erreur à la compilation car la variable i n'est pas déclarée (la portée de la variable i déclarée en ligne 12 est limité à la boucle)
5. La variable f pointe vers l'adresse de a ; la valeur de a en ligne 21 passe de 100 à 200 ; on a donc *f = 200.

   Par contre, g pointe vers une variable entière stockée sur le tas, initialisée à 100 (la valeur de *f à la ligne 20), et pas affectée par la ligne 21. On a donc *g=100.
6. On a &h[1]+2=&h[3] (adresse du quatrième élément de h) et (&h[3])[-1]=h[2]=5. On a donc m=h[5]=11
7. En ligne 27, le tableau de char h est converti en pointeur vers int, puis tableau de int, dont le premier élément est renvoyé.

   En IA32 un int prend 4 octets en mémoire, alors qu'un char n'en prend qu'un.

La valeur de n est l'entier dont le bitstring correspond aux quatre premières entrées de h. (De la même façon, ((int*)h)[1] correspondrait aux quatre entrées suivantes, etc.)

La bonne réponse est une autre valeur.

8. Pas d'erreur à la compilation.

   En ligne 31, la variable o pointe vers une zone mémoire qui a été désallouée (en ligne 30). Son contenu est donc indéterminé, mais pas d'erreur a priori à l'exécution.

9. La transmission des paramètres se faisant par copie, ceux-ci ne sont pas affectés par le corps de la fonction. Le programme renvoie donc x = 100 y = 1000 en ligne 40.