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 — août 2024

- 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).
- Ecrivez de façon lisible...

Question 1

Soit le code suivant

```
1  %include "io.inc"
2  CPU 386
3  global CMAIN
4
5  section .data
6  LISTE DW 642, 21512, 1030, 263, 16393, 32770, 10248, 1322
7  LEN_LISTE DB 8
8  SIZE DB 16
9
10 section .text
11 CMAIN:
12     XOR eax, eax
13     MOVZX edx, BYTE[LEN_LISTE]
14 bcle:
15     MOV ecx, [SIZE]
16     MOVZX ebx, WORD[LISTE+2*(edx-1)]
17 bcle2:
18     SHR bx, 1
19     ADC eax, 0
20     LOOP bcle2
21     DEC edx
22     CMP edx, 0
23     JA bcle
24     PRINT_UDEC 4, eax
25     RET
```

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

642	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0
21512	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0
1030	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
263	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1
16393	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
32770	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
10248	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0
1322	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0

On vous demande :

1. d'expliquer le code de la ligne **15** à ligne **20**. Bien expliquer les assignations et l'impact des instructions sur les registres.

   [3 points]

2. d'indiquer à la ligne **18** : `SHR bx, 1` si on peut remplacer l'instruction par
   — `SHL bx, 1`
   — `RCL bx, 1`
   — `ROR bx, 1`
   — `SHL ebx, 1`
   — `RCL ebx, 1`
   — `ROR ebx, 1`
   pour que le code fonctionne toujours correctement. Si c'est le cas, indiquez aussi ce que `bx` ou `ebx` vaudra après la ligne **20** : `LOOP bcl2`. Considérez uniquement les 3 valeurs possibles :
   — 0
   — la valeur initiale de `bx` ou `ebx`
   — une autre valeur

   [3 points]

3. d'expliquer brièvement et pour une personne n'ayant que des connaissances basiques des systèmes binaires ce que fait le code assembleur et d'indiquer ce que le programme imprime sur l'output

   [3 points]

4. d'indiquer les changements à apporter si on veut traiter une liste de nombres codés sur 32 bits

   [1 point]

Question 2

Répondez aux questions ci-dessous sur base du code C++ et de Makefile donnés à la page suivante. Justifiez *brièvement* vos réponses.

1. Le code compile-t-il si la ligne **1** est retirée du fichier `fct.cpp` ? Pourquoi ?
   [2 points]
2. Le code compile-t-il si la ligne **2** est retirée du fichier `main.cpp` ? Pourquoi ?
   [2 points]
3. Dans le fichier `main.cpp` :
   (a) Qu'affiche le programme lors de l'exécution de la ligne **7** ?
   [2 points]
   (b) Qu'affiche le programme lors de l'exécution de la ligne **8** ?
   [1 point]
   (c) Qu'affiche le programme lors de l'exécution de la ligne **9** ?
   [1 point]
4. Dans le Makefile fourni, à quoi correspondent les symboles `$<`, `$^` et `$@` ?
   [2 points]

Listing 1 – Makefile

```
FLAGS = -Wall -Wextra
prog: main.o fct.o
	g++ $^ -o $@
%.o: %.cpp
	g++ -c $(FLAGS) $< -o $@
```

Listing 2 – main.cpp

```
#include "fct.hpp"
extern int z;
int main() {
    int a = 0, b = 10, c = 100;
    a = f1(a);
    c = f1(c);
    std::cout << a << ", " << c << std::endl;
    std::cout << f2(b) << std::endl;
    std::cout << f3(5, 2) << std::endl;
    return z;
}
```

Listing 3 – fct.cpp

```
#include "fct.hpp"
int f1(int& x) {
    static int& y = x;
    return ++y;
}
int f2(int m) {
    int ret = 0;
    while(--m > 0)
        ret++;
    return ret;
}
float f3(int a, int b) {
    return a / b;
}
int z = 0;
```

Listing 4 – fct.hpp

```
#include <iostream>

int f1(int&);
int f2(int);
float f3(int, int);
```

Question 3

En raison d'une panne mondiale ayant perturbé les Jeux Olympiques du Web (JOW), vous êtes appelé d'urgence en tant qu'étudiant en informatique pour développer un système de gestion de crise. Votre tâche est de créer un outil pour organiser les ressources, le personnel et gérer les événements.

L'utilisation de librairies externes est permise tant que cela est justifié.

Justifiez l'utilisation des mots-clés *privé* et *public*. Expliquez pourquoi certains attributs et méthodes sont privés et d'autres sont publics.

1. Implémentez une structure **Volunteer** avec :
   — un attribut **name** pour le nom du volontaire.
   — un attribut **skill** pour la compétence principale du volontaire.
   — un attribut **availableHours** pour le nombre d'heures de disponibilité du volontaire.

   [2 points]

2. Implémentez une classe **CrisisTeam** avec :
   — un attribut **location** indiquant l'emplacement de l'équipe de crise.
   — un attribut **members**, un vecteur qui stockera les volontaires recrutés (sans limite de taille).
   — une méthode **addMember** pour ajouter un volontaire à l'équipe.
   — une méthode **totalAvailableHours()** qui retourne le total des heures disponibles de tous les membres.

   [3 points]

3. Implémentez les surcharges d'opérateur suivantes :
   (a) Opérateur + pour combiner deux équipes.
   Cette opération doit créer une nouvelle équipe contenant tous les membres des deux équipes, avec une nouvelle localisation "combined location".
   (b) Opérateur += pour ajouter un volontaire à une équipe.
   (c) Opérateur −= pour retirer un volontaire d'une équipe (par son nom).

   [3 points]

4. Implémentez les surcharges d'opérateur pour l'affichage :
   (a) `ostream& operator«(ostream& os, const Volunteer& volunteer)`, qui affichera une instance de **Volunteer** sous le format : `Nom (Compétence, Heures disponibles)`.
   (b) `ostream& operator«(ostream& os, const CrisisTeam& team)`, qui affichera une instance de **CrisisTeam**. Le format d'affichage est donné dans l'exemple d'utilisation ci-dessous.

   [2 points]

Exemple d'utilisation :

```
int main() {
    Volunteer v1 = {"Alice", "First Aid", 20};
    Volunteer v2 = {"Bob", "Security", 15};
    Volunteer v3 = {"Charlie", "Logistics", 25};
    Volunteer v4 = {"David", "IT Support", 18};

    CrisisTeam olympicVillage("Olympic Village");
    CrisisTeam stadium("Main Stadium");

    olympicVillage += v1;
    olympicVillage += v2;
    stadium += v3;
    stadium += v4;

    cout << olympicVillage << endl;
    cout << stadium << endl;

    CrisisTeam combinedTeam = olympicVillage + stadium;
    cout << combinedTeam << endl;

    combinedTeam -= "Bob";
    combinedTeam.sortByAvailability();
    cout << "After removing Bob and sorting:" << endl;
    cout << combinedTeam << endl;

    return 0;
}
```

Affichera ceci (4 lignes affichées) :

Crisis team at Olympic Village with 2 members (35 total hours) : Alice (First Aid, 20) Bob (Security, 15)

Crisis team at Main Stadium with 2 members (43 total hours) : Charlie (Logistics, 25) David (IT Support, 18)

Crisis team at Combined Team with 4 members (78 total hours) : Alice (First Aid, 20) Bob (Security, 15) Charlie (Logistics, 25) David (IT Support, 18)

After removing Bob and sorting : Crisis team at Combined Team with 3 members (63 total hours) : Charlie (Logistics, 25) Alice (First Aid, 20) David (IT Support, 18)

Question 4

Considérez le programme C++ suivant.

```
1  #include <iostream>
2  using namespace std ;
3
4  int main() {
5      int a = 1 ;
6      int* b = new int(a) ;
7      int c = *(b) ;
8      int& d = c;
9      int* e = b;
10
11     //a++;
12     //b++;
13     //(*b)++;
14     //c++;
15     //d++;
16     cout<<"a="<<a<<" *b="<<*b<<" c="<<c<<endl;
17     cout<<" d="<<d<<" *e="<<*e<<endl;
18     //delete e ;
19
20     char f = 16;
21     char g = f*f ;
22     int diff=g-f*f;
23     cout<<"diff="<<diff<<endl;
24
25     int tabint[12]={1,2,4,8,16,32,64,128,256,512,1024,2048};
26     int h = *(tabint+3)*((&tabint[2]+1)[1]);
27     cout<<"h="<<h<<endl;
28     //char i = tabint[12];
29
30     union U {int i; char c;} ;
31     U tabu[12];
32     for(int ind=0;ind<12;ind++){
33         if(ind%2) {(tabu[ind]).i=tabint[ind];}
34         else {(tabu[ind]).c=tabint[ind];}
35         cout<<(tabu[ind]).i<<endl;
36     }
37
38     return 0;
39 }
```

Répondez aux questions suivantes, **en justifiant toutes vos réponses**.

1. Si on décommente la ligne 11, qu'est-ce que le programme affichera en lignes 16 et 17 ? Justifiez votre réponse.

   [1 point]

2. Si on décommente la ligne 12, qu'est-ce que le programme affichera en lignes 16 et 17 ? Justifiez votre réponse.

   [1 point]

3. Si on décommente la ligne 13, qu'est-ce que le programme affichera en lignes 16 et 17 ? Justifiez votre réponse.

   [1 point]

4. Si on décommente la ligne 14, qu'est-ce que le programme affichera en lignes 16 et 17 ? Justifiez votre réponse.

   [1 point]

5. Si on décommente la ligne 15, qu'est-ce que le programme affichera en lignes 16 et 17 ? Justifiez votre réponse.

   [1 point]

6. Si on décommente la ligne 18, le programme produira-t-il une erreur à la compilation et/ou à l'exécution ? Justifiez vos réponses.

   [1 point]

7. Qu'est-ce que le programme affichera en ligne 27 ? Justifiez votre réponse.

   [1 point]

8. Si on décommente la ligne 28, 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 35 ? Justifiez vos réponses.

   [1 point]

10. A la ligne 22, le programme affiche *diff=-256*. Donnez une explication plausible de ce résultat.

    [1 point]

Examen INFO-F105 Langages de Programmation 1 ; examen – août 2024

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

Numéro de question : 1

---

1. Explication des instructions
   — ligne **15** : on assigne `ecx` à la valeur de `SIZE`, soit 16, la taille des éléments de la liste `LISTE` ;
   — ligne **16** : on assigne `ebx` à l'élément de la liste `LISTE` en partant du dernier élément. L'initilisation de `edx` étant ligne **13** et le décrémentation est à la ligne **21**
   — ligne **17** : contient un label `bcle2`
   — ligne **18** : le `SHR` de `bx` décale tous les bits de `bx` vers la droite, le bit de poids fort de `bx` passe à 0 et le bit de poids faible tombe dans de `CF`, le *Carry Flag*
   — ligne **19** : l'instruction de `ADC`, ajoute 0 **et le `CF`** à `eax`.
   — ligne **20** : l'instruction `LOOP` décrémente `ecx` et si `ecx` ne tombe pas à 0, l'instruction fait un *jump* vers le label `bcle`, ligne **17**.
2. Remplacer `SHR bx, 1`
   — `SHL bx, 1` : OK, `bx` vaudra 0 après les 16 itérations
   — `RCL bx, 1` : OK, mais `bx` contiendra une valeur modifiée après 16 itérations compte tenu de `RCL` et de la valeur initiale du `CF`
   — `ROR bx, 1` : OK et `bx` contiendra toujours sa valeur initiale après les 16 itérations
   — `SHL ebx, 1` : NON, les nombres sont sur 16 bits, `ebx` est sur 32 bits
   — `RCL ebx, 1` : NON, les nombres sont sur 16 bits, `ebx` est sur 32 bits
   — `ROR ebx, 1` : OK, mais `bx`, après 16 itérations, sera la valeur initiale $\times 2^{16}$
3. Le code somme les bits à 1 des nombres en base 2 de la liste. Le code imprimera 27 sur l'output.
4.
```
LISTE DD .... ; ligne 6
SIZE DB 32 ; ligne 8
mov ebx, [ LISTE + 4 * (edx - 1) ] ; ligne 16
shr ebx , 1 ; ligne 18
```

Examen INFO-F105 Langages de Programmation 1 ; examen – août 2024

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

Numéro de question : 2

---

1. Oui le code compile sans problème : les fonctions sont définies mais jamais appelées dans ce fichier. Il n'y a donc pas besoin des prototypes.
2. Non : la variable `z` est une variable globale définie dans `fct.cpp`. Pour pouvoir être utilisée dans un autre fichier, il faut impérativement signaler au compilateur que cette variable est définie dans un autre fichier à l'aide du mot-clef `extern`.
3. (a) La référence locale `f1::y` est *statique*, donc n'est initialisée qu'une seule fois, lors du premier appel. Le premier appel `f1(a)` va donc initialiser `f1::y` à `main::a` et puis va incrémenter cette valeur et finalement la renvoyer. Elle vaut maintenant 1.

   Lors du second appel, cette même valeur va être incrémentée une nouvelle fois et renvoyée une nouvelle fois. Elle vaut maintenant 2. Cependant `f1::y` est une *référence* vers `main::a`, donc à la fin `main::a` et `main::c` valent toutes les deux 2. Il sera donc affiché « `2, 2` ».

   (b) La fonction `f2` décrémente le paramètre reçu jusqu'à atteindre 0 et à chaque tour de boucle la variable locale `ret` est incrémentée.

   Attention cependant : l'incrément est un *pré-incrément* et donc `ret` ne va être incrémentée que `m-1` fois puisqu'au mème tour de boucle, le préincrément va mettre `m` à 0 et puis seulement exécuter la comparaison.

   Il sera donc affiché « 9 ».

   (c) Il sera affiché « 2 »puisque `f3` effectue une division *entière* : la division de deux entiers donne un entier.
4. — Le symbole `$<` désigne la première dépendance d'une target ;
   — le symbole `$^` désigne l'entièreté des dépendances d'une target ;
   — le symbole `$@` désigne le nom de la target.

Examen INFO-F105 Langages de Programmation 1 ; examen – août 2024

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

Numéro de question : 3

```
1
2
3
4  #include <iostream>
5  #include <string>
6  using namespace std;
7
8  const int N_a_Members = 100;  // Taille maximale du tableau
   ↪ des membres
9
10 struct Volunteer {
11     string name;
12     string skill;
13     unsigned int availableHours;
14 };
15
16 class CrisisTeam {
17 private:
18     string location;
19     Volunteer members[N_a_Members];
20     unsigned int counter; // Compteur pour suivre le nombre
       ↪ de membres ajoutés
21
22 public:
23     // Constructeur pour initialiser l'emplacement et le
       ↪ compteur
24     CrisisTeam(string loc) : location(loc), counter(0) {}
25
26
27     void addMember(const Volunteer& volunteer) {
28         if (counter < N_a_Members) {
29             members[counter++] = volunteer;
30         } else {
31             cout << "Team is full." << endl;
32         }
33     }
34
35
```

```
36      int totalAvailableHours() const {
37          unsigned int total = 0;
38          for (int i = 0; i < counter; ++i) {
39              total += members[i].availableHours;
40          }
41          return total;
42      }
43
44
45      friend CrisisTeam operator+(const CrisisTeam& team1,
            ↪ const CrisisTeam& team2) {
46          CrisisTeam combinedTeam("Combined Location");
47          for (int i = 0; i < team1.counter; ++i) {
48              combinedTeam.addMember(team1.members[i]);
49          }
50          for (int i = 0; i < team2.counter; ++i) {
51              combinedTeam.addMember(team2.members[i]);
52          }
53          return combinedTeam;
54      }
55
56
57      friend CrisisTeam& operator+=(CrisisTeam& team, const
            ↪ Volunteer& volunteer) {
58          team.addMember(volunteer);
59          return team;
60      }
61
62      friend CrisisTeam& operator-=(CrisisTeam& team, const
            ↪ string& volunteerName) {
63          for (int i = 0; i < team.counter; ++i) {
64              if (team.members[i].name == volunteerName) {
65                  for (int j = i; j < team.counter - 1; ++j) {
66                      team.members[j] = team.members[j + 1];
67                  }
68                  --team.counter;
69                  break;
70              }
71          }
72          return team;
73      }
74
75
76      friend ostream& operator<<(ostream& os, const Volunteer&
            ↪  volunteer) {
77          os << volunteer.name << " (" << volunteer.skill << "
```

```
            ↪ , " << volunteer.availableHours << ")";
78          return os;
79      }
80
81      friend ostream& operator<<(ostream& os, const CrisisTeam
            ↪ & team) {
82          os << "Crisis team at " << team.location << " with "
                ↪  << team.counter << " members ("
83              << team.totalAvailableHours() << " total hours):
                    ↪ ";
84          for (int i = 0; i < team.counter; ++i) {
85              os << team.members[i] << " ";
86          }
87          return os;
88      }
89  };
```

Examen INFO-F105 Langages de Programmation 1 ; examen – août 2024

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

Numéro de question : 4

---

1. Le programme affiche *a=2 *b=1 c=1 d=1 *e=1.*

   En ligne 5, le programme a créé une variable a sur la pile et l'a initialisée à 1. En ligne 6, le programme a alloué de l'espace sur le tas pour une variable entière, a initialisée celle-ci à 1 (la valeur de a), et a stocké son adresse dans un pointeur b sur la pile. En ligne 7, le programme a créé une nouvelle variable c sur la pile, et l'a initialisée à 1 (la valeur de la mémoire pointée par b). En ligne 8, le programme a créé une variable référence d, qui agira comme un alias de c. En ligne 0, le programme a créé un pointeur e, et lui a donné comme valeur la valeur de b, càd l'adresse de la variable sur le tas.

   En ligne 11, on incrémente simplement la valeur de a. Ceci n'a aucun effet sur les autres variables car elles ne sont pas liées à a.

2. Ce que le programme affichera est INDEFINI. Plus précisément, le programme affichera *a=1 *b=X c=1 d=1 *e=1*, où la valeur X est indéfinie.

   Suite à l'incrément, b pointe maintenant vers l'adresse suivante sur le tas, dont la valeur n'est pas définie (a priori sa valeur vaudra 0, mais ce n'est pas garanti). Les autres variables ne sont pas modifiées.

3. Le programme affiche *a=1 *b=2 c=1 d=1 *e=2.*

   On incrémente ici la variable pointée par b et non pas son adresse. On note aussi que e pointe vers la même zone mémoire, et que les autres variables ne sont pas affectées par le changement.

4. Le programme affiche *a=1 *b=1 c=2 d=2 *e=1.*

   On incrémente la variable c, dont d est un alias (référence vers c). Les autres variables ne sont pas affectées par le changement.

5. Le programme affiche *a=1 *b=1 c=2 d=2 *e=1.*

   De la même façon, on incrémente d, qui est un alias de c. Les autres variables ne sont pas affectées par le changement.

6. Il n'y aura pas d'erreur à la compilation (l'opérateur delete s'applique à un pointeur, ce qui est correct) ni à l'exécution (le pointeur est une zone de mémoire allouée sur le tas).

   On peut remarquer que cette instruction laisse les pointeurs b et e en l'état ; une bonne pratique serait de les réinitialiser à zéro pour éviter d'accéder à une zone mémoire qui a été désallouée. Néanmoins, dans le cas du programme ici, ni b ni e ne sont utilisés au-delà de la ligne 18, et aucune erreur ne se produira.

7. Le programme affiche *h=128.*

L'expression se décompose comme un produit de deux opérandes. Le premier opérande *(tabint+3) vaut la valeur de tabint[3] c'est-à-dire 8. Le deuxième opérande vaut la valeur de tabint[4] c'est-à-dire 16. Le produit de ces deux opérandes est 8*16=128.

8. Pas d'erreur à la compilation ; l'exécution pourrait en théorie produire une erreur même si a priori non.

   Le programme va aller lire la zone mémoire juste après le tableau tabint ; son contenu n'est pas défini mais il est a priori accessible, car tabint est sur la pile. Donc a priori pas d'erreur.

   Si tout se passe bien jusque-là, cette zone mémoire (interprétée comme un int) va être convertie en char (avec perte d'information correspondante). Ceci est tout-à-fait conforme au langage, donc pas d'erreur.

   Notons aussi que une autre variable de nom i est déclarée n ligne 30 (un attribut de U) ; ceci est permis car la portée de celle-ci est locale.

   Notons enfin que une opération comme tabint[1000] est beaucoup plus susceptible d'amener une erreur à l'exécution, car on risque d'essayer d'accéder à la zone kernel, non accessible au programme. Par ailleurs, l'organisation de la mémoire telle que vue au cours est généralisée en pratique mais pas imposée par le standard ; en théorie, on pourrait très bien avoir une zone mémoire inaccessible au programme qui suit directement tabint.

9. Le résultat est INDEFINI.

   Plus précisément, les valeurs d'indice pair de tabu sont indéfinies car on a stocké ces valeurs comme des charactères, pour ensuite les lire et les imprimer comme des entiers. Leurs valeurs dépendent donc de la représentation en mémoire, mais aussi potentiellement de l'état de la mémoire avant la création du tableau.

   Les valeurs d'indice impair sont elles bien définies : 2,8, 32, 128, 512, 2048.

   Pour les valeurs d'indice pair, les valeurs stockées (comme char) seront 1, 4, 16, 64, 0, 0. Imprimer ces valeurs comme int pourrait produire les mêmes entiers, selon la représentation de l'union en mémoire et l'état des bits avant utilisation.

10. Contrairement à ce qu'on pourrait croire a priori, le résultat de la multiplication de deux variables de types char n'est pas une variable de type char, mais une variable de type int. Plus précisément, la multiplication de deux variables de type char n'est pas directement définie par le langage, mais elle prend sens à travers les conversions implicites et la mulitplication de deux variables de type int. Les opérandes char sont implicitement converties en int, puis l'opération est effectuée sur les deux variables int.

    L'évaluation de l'expression f*f produit donc une variable temporaire de type int et de valeur $16 * 16 = 256$.

    En ligne 21, cette valeur est convertie en un char, qui se représente sur 8 bits. Il y a donc dépassement d'entier, et la valeur de g est 0.

    En ligne 22, la variable g (de valeur 0) est convertie en int (pour donner une variable temporaire de valeur 0), puisque l'autre opérande de l'opérateur - est une variable temporaire de type int.

    La différence est donc bien un entier de valeur 0-256=-256.

Notez que je ne m'attends pas à une justification complète pour cette question, car celle-ci est subtile et avancée. J'ai accordé des points de façon généreuse pour des débuts d'intuition dans la bonne direction.

De façon générale, les points accordés portaient davantage sur les justifications que sur les réponses numériques ou binaires.