LOG1000 - Ingénierie logicielle
Hiver 2013
Examen final

Enseignant : Julien Gascon-Samson

Coordonnateur : Bram Adams

Cet examen comporte 6 questions pour un total de 40 points

Pondération : 40% de la note finale

Directives :
Toute documentation permise.
Répondez sur le questionnaire et remettez-le à la fin de l'examen.

1 Questions générales (5 points)

Pour les 5 questions suivantes, choisissez la réponse qui vous semble être la bonne et indiquez votre choix **en remplissant le tableau au début de la page 5** (0,5 point par question).

1.1 Un test de régression

a) est un nom que l'on utilise pour dénommer n'importe quel test.

b) est le nom que l'on utilise pour des tests qui sont écrits après le codage d'une classe, contrairement aux tests écrits avant le codage d'une classe par des approches agiles.

c) est un test qui a pour but de vérifier si le système implémente toutes les fonctionnalités requises par le client.

d) RÉPONSE■ est un test qui a pour but de vérifier si les bogues qui ont été résolus avant ne réapparaissent plus jamais après.

e) est un test utilisé lorsque l'on suit la programmation défensive. Les tests utilisés avec la programmation offensive sont appelés des tests d'aggression.

1.2 Jean-Rémi développe une application web avec Scala qui doit prendre un matricule et le stocker dans une base de données. Quelle serait la meilleure stratégie de tests ?

a) Écrire un test avec son propre matricule pour tester le scénario principal.

b) Ignorer les tests, car Scala est si avancé que les développeurs ne font pas de bêtises.

c) Écrire un test pour simuler des dizaines de personnes en même temps en train d'accéder à l'application.

d) RÉPONSE■ Écrire un test pour des cas de bord comme des matricules malformés ou des matricules vides.

e) Écrire un test pour simuler que le disque dur est plein.

1.3 Dans une équipe de 13 personnes, il y a en théorie X canaux de communication :

a) RÉPONSE- 78

b) 6,227,020,800

c) 156

d) 13

e) 26

1.4 La différence entre la portée et la durée de vie d'une variable est :

a) inexistante, ce sont deux mots pour le même concept.

b) RÉPONSE- la différence entre théorie et pratique, c.-à-d. la région maximale de code (par exemple une fonction) où une variable peut exister en théorie versus la région maximale dans laquelle elle est utilisée.

c) la différence entre pratique et théorie, c.-à-d. la région maximale dans laquelle une variable est utilisée versus la région maximale de code (par exemple une fonction) où elle peut exister en théorie.

d) liée au problème des variables allouées sur le tas (“heap”) qui sont encore utilisées dans le code après être désallouées.

e) très petite. La portée est la distance entre chacune des deux références d’une variable, tandis que la durée de vie d’une variable est la distance entre la première et la dernière référence d’une variable.

1.5. Quel type de cohésion le programme suivant contient-il :

```
1  void completeData(std::vector<std::string>& data) {
2    //convert to lowercase
3    for(std::vector<std::string>::iterator it=data.begin();it!=data.end();it++){
4      std::transform(it->begin(), it->end(), it->begin(), std::tolower);
5    }
6
7    //replace numbers by X
8    for(std::vector<std::string>::iterator it=data.begin();it!=data.end();it++){
9      int pos=0;
10     do{
11       pos = it->find_first_of("0123456789", pos);
12       if(pos != std::string::npos){
13         it->replace(pos,1,"X");
14       }
15     }while(pos != std::string::npos);
16   }
17
18   //encrypt
19   for(std::vector<std::string>::iterator it=data.begin();it!=data.end();it++){
20     *it=encrypt(*it);
21   }
22 }
```

a) Cohésion temporelle

b) RÉPONSE Cohésion de communication

c) Cohésion de contrôle

d) Cohésion fonctionnelle

e) Cohésion procédurale

Pour les questions 1 à 5, écrivez lisiblement la lettre corrrespondant à votre réponse.

1	2	3	4	5
d	d	a	b	b

Pour chaque énoncé qui suit, dites s'il est vrai ou faux (une bonne réponse vaut 0,5 point et une mauvaise réponse vaut -0,25 point).

		Vrai	Faux
1.6	Un test de système requiert moins d'effort qu'un test d'installation.		X
1.7	Ajouter une commande *print* est une restructuration.		X
1.8	Une classe en C++ est une implémentation d'un type abstrait de données.	X	
1.9	Il vaut mieux optimiser la performance dès la première version d'une classe.		X
1.10	En pratique, les tests du flux de données remplacent les tests de base structurés, car ils sont plus précis.		X

2 Code de qualité (8 points)

Répondez aux 5 questions suivantes (5 points).

a) Expliquez pourquoi l'énoncé qui suit exprime un avantage des types abstraits de données : "Les changements au type de données n'affectent pas tout le programme". (1 point) Masquage d'information.

b) Donnez l'interface d'une classe C++ pour une télévision. Limitez-vous aux 5 fonctionnalités les plus utilisées. (1 point) allumer, éteindre, changer de channeau, changer le volume du son, start recording, ...

c) Donnez cinq noms de patrons de conception. (1 point) observer, singleton, factory method, mediator, bridge, template method, ...

d) Pourquoi dit-on que des méthodes peuvent "briser l'enchaînement multiple de boucles" ? Donnez un exemple en C++. (1 point) Extract nested loop into method.

e) Expliquez deux rôles du nombre "7" souvent utilisé dans la programmation/le code, et expliquez le raisonnement. (1 point) Maximum number of things people can keep in their short-term memory. Maximum number of methods in interface, arguments, attributes, ...

Pour les deux questions qui suivent, considérez le code fourni **en annexe A** (3 points).

f) Calculez le *span moyen* de chacune des variables de la fonction *Seeker_FindEnemy* (attention : seulement la réponse sera évaluée). Comptez les lignes de commentaires, mais **pas** les lignes vides ou les lignes contenant seulement *{* ou *}*. Si le résultat n'est pas un entier, laissez-le sous forme de fraction simplifiée. Ignorez les variables globales *gi* et *NPC*, ainsi que les variables locales *ent* et *i*. (1,5 point)
span(numFound)=(5+0)/2=5/2
span(dis)=(17+0+0)/3=17/3
span(bestDis)=(18+0)/2=9
span(mins)=(2+0)/2=1
span(maxs)=(1+0+0)/3=1/3
span(entityList)=(2+1)/2=3/2
span(best)=(18+0+3)/3=7

g) Comment pourrait-on modifier la déclaration des variables afin de minimiser le span ? Pour chaque variable, indiquez **avant** quelle ligne elle devrait être déclarée, et recalculez les *spans moyens*. Traitez chaque cas indépendamment. Exemple : "déclarer x **avant la ligne** 11". (1,5 point)
numFound : before line 11 => (0+0)/2=0
dis : before line 36 => (0+0+0)/3=0
bestDis : before line 13 => (13+0)/2=13/2

mins : before line 9 => (0+0)/2=0
maxs : before line 8 => (0+0+0)/3=0
entityList : before line 11 => (0+1)/2=1/2
best : before line 13 => (15+0+3)/3=6

3 Débogage (4 points)

Considérez la classe `list` de l'exemple `ListInsert` vue en cours. Cette classe encapsule une liste d'entiers [1]. En tout temps, les éléments de la liste sont triés en ordre croissant. La classe fournit deux méthodes :

- `insert`, prenant en paramètre un entier (`newY`), permettant d'insérer l'élément `newY` dans la liste de façon à maintenir le tri en ordre croissant
- `getElement`, prenant en paramètre un index (`index`), permettant d'obtenir l'élément à la position `index`.

La classe `ListTest` contient six cas de tests unitaires exécutés à l'aide du cadriciel `cppunit`. Puisque le code de la classe `list` contient des erreurs, nous avons décidé d'utiliser le débogueur de l'environnement de développement Visual Studio afin de faciliter notre effort de recherche et de correction d'erreurs. La capture-écran 1 illustre une vue de l'environnement en cours de débogage [2].

Remarque importante : l'implémentation de la classe `list` est donnée en annexe B et l'implémentation de la classe `ListTest` est donnée en annexe C.

Répondez aux questions suivantes :

a) Parmi les six cas de tests, quel cas de test est présentement en cours d'exécution dans la capture-écran 1 ? Justifiez votre réponse. (0,75 pt)

Test 2, car l'appel à la méthode test2 est indiquée dans la pile (stack)

1. Maximum 10 éléments.
2. Les numéros de lignes de la capture-écran correspondent avec les numéros de lignes des listings de code.

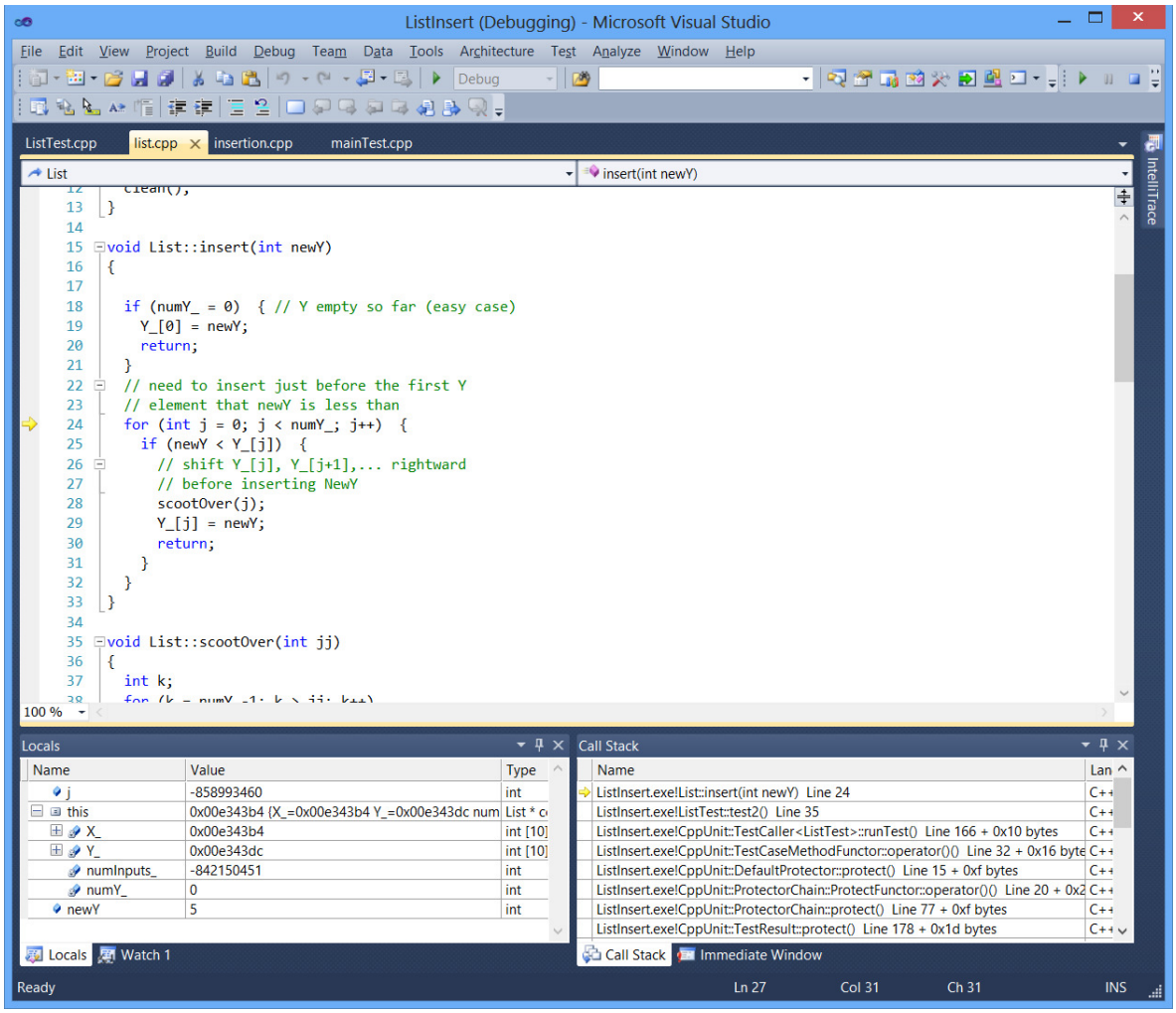


FIGURE 1 – Débogage avec Visual Studio (1)

b) En vous fiant au code source du cas de test indiqué en (a) ainsi qu'à la capture-écran 1, quel problème remarquez-vous avec le **curseur d'exécution** ? (0,75 pt)

Si le nombre d'éléments dans la liste est 0 (`numY_==0`), le curseur aurait dû passer par la ligne 20 et par conséquent ne jamais atteindre la ligne 24 !

c) Proposez une correction qui permettrait de **régler le problème mentionné en (b)**. Indiquez à quel emplacement il faut effectuer un changement et indiquez les modifications effectuées. (1 pt)

Il faudrait modifier la ligne 18 : `if (numY_ == 0)`

d) Après avoir effectué les changements décrits en (c), lorsque nous réexécutons **le même cas de test**, le flot d'exécution passe par la ligne 82 du fichier `list.cpp`. Indiquez pourquoi cela ne fait pas de sens et proposez une autre correction à effectuer (indiquez l'emplacement de la correction et le code modifié). (1,5 pt)

Après l'insertion d'un item, la clause `index >= numY_` devrait évaluer à faux. Or, elle évalue à vraie, ce qui provoque l'exception. La variable `numY_` n'a pas été incrémentée : il faut rajouter `numY_++` ; entre les lignes 19 et 20.

4 Gestion de construction du code (3 points)

a) Un développeur incompétent aura-t-il une influence négative plus grande sur un projet logiciel de grande taille ou sur un projet de petite taille ? Justifiez votre réponse. (1 point)

Sur un projet de petite taille, car le succès du projet reposera davantage sur la compétence individuelle des programmeurs.

b) Dessinez un graphe du nombre moyen d'erreurs par kilo-ligne de code (densité d'erreurs) en fonction de la taille du projet. Expliquez votre raisonnement dans le cas d'un grand projet. (1 point)

Réponse 1 : davantage erreurs proviennent d'autres activités : spécifications, architecture, conception, tests... Réponse 2 : dans un grand projet, il y a beaucoup d'interdépendances entre les modules. Lorsqu'on ajoute du nouveau code, notre code risque de dépendre d'un nombre plus élévé de modules, ce qui augmente les possibilités d'erreurs.

c) Il n'est jamais trivial d'ajouter des développeurs en cours de projet. Il est toutefois possible de prendre certaines mesures afin de faciliter l'ajout de nouveaux membres. Quelle stratégie pourrait-on adopter au moment de la conception afin de faciliter l'intégration de nouveaux membres à l'équipe de développement ? (1 point)

Diviser le code en modules distincts bien découplés (architecture modulaire) : de cette façon, on pourra assigner des modules spécifiques aux nouveaux développeurs. Ils n'auront pas à comprendre tout le code comme ce serait le cas avec un code monolithique.

5 Processus de programmation en pseudocode (3 points)

Soit une méthode `int obtenirPairs(int elements[], int nombre, int elementsPairs[])` qui prend en paramètre un tableau d'entiers (`elements`) ainsi que le nombre d'éléments contenus dans le tableau `elements`. Cette méthode ajoute l'ensemble des nombres du tableau `elements` qui sont pairs au tableau `elementsPairs` et retourne le nombre d'éléments pairs trouvés (qui correspondra au nombre d'éléments retournés dans le tableau `elementsPairs`)[3].

En cours, nous avons vu le processus de programmation en pseudocode. Utilisez cette technique afin d'implémenter la méthode `obtenirPairs`. **Utilisez la méthodologie décrite en cours !** Procédez en trois étapes : (a) pseudocode au niveau des intentions, (b) pseudocode détaillé et (c) transformez le pseudocode en code (C/C++ sans utiliser la STL).

a) Pseudocode au niveau des intentions (0,5 point)

```
int obtenirPairs(int elements[], int nombre, int elementsPairs[])
{




}
```

3. Considérez que le tableau `elementsPairs` possède la même taille que le tableau `elements`

b) Pseudocode détaillé (1 point)

```
int obtenirPairs(int elements[], int nombre, int elementsPairs[])
{









}
```

c) Pseudocode transformé en code (C/C++ sans utiliser la STL) (1,5 point)

```
int obtenirPairs(int elements[], int nombre, int elementsPairs[])
{









}
```

6 Programmation défensive (4 points)

a) Dans un bon nombre d'applications à interface graphique, lorsqu'une erreur survient, un message apparaît à l'utilisateur pour l'avertir du problème rencontré. Cette approche possède certains avantages, mais aussi des inconvénients. Nommez deux inconvénients. (1 point)

Mauvais niveau d'abstraction pour l'usager : infos techniques possiblement pas pertinentes. Risque que le message interfère avec le bon fonctionnement de l'interface. Du point de vue du code, trop grand couplage vers interface usager / non-respect du principe MVC.

b) Considérez l'extrait de code qui suit [4] :

```
1  int main (int argc, char* argv[])
2  {
3    ASSERT(argc>1);
4    ASSERT(argv != NULL);
5
6    FILE* pFile;
7    ASSERT(pFile = fopen (argv[1],"w"));
8
9    fputs ("fopen example",pFile);
10
11   % Si la fermeture du fichier échoue, fclose retournera EOF
12   int retour = fclose (pFile);
13   ASSERT(retour != EOF);
14
15   return 0;
16 }
```

Cet extrait illustre-t-il une bonne utilisation des assertions ? Pour chaque assertion, justifiez si l'utilisation est appropriée ou non. (2 points)

1. Non, car le nombre de paramètres provient de la commande entrée par l'usager : un traitement d'erreur est approprié (assertion supprimée en production).
2. Oui, car ne devrait jamais se produire. Si cela arrive, il s'agit d'un problème exceptionnel.
3. Non, car le code d'ouverture du fichier sera supprimé en production.

4. `argc` est le nombre d'arguments passés en paramètres. `argv` est un tableau contenant : le chemin vers l'exécutable (élément 0) suivi des arguments passés en paramètres.

4. Les deux cas acceptables. Non car c'est exceptionnel. Oui car une telle erreur pourrait théoriquement survenir. Justifier !

c) Différents mécanismes existent pour signaler une situation d'erreur. Parmi les techniques les plus connues, nous avons vu en cours les exceptions ainsi que le retour d'un code d'erreur [5]. Mentionnez deux avantages découlant de l'utilisation des exceptions par rapport au retour d'un code d'erreur. (1 point)

1. Meilleure abstraction de la sous-routine : pas besoin de "polluer" la valeur de retour avec des codes de retour (demande une connaissance interne de la méthode / moins de connaissances sémantiques à savoir).
2. Si c'est une méthode (sans valeur de retour) : pas besoin de transformer une méthode en fonction juste pour avoir un code de retour.
3. Possibilité de traiter l'erreur à différents niveaux dans la hiérarchie d'appels.

5. Une sous-routine retourne un code spécifique pour signaler une erreur (exemple : NULL, -1, EOF, etc.)

7 Restructuration du code (6 points)

Les exemples de code suivants contiennent des *mauvaises odeurs*. Pour chaque exemple :

- Indiquez en une ligne quel est le principal problème [6].
- Décrivez brièvement la restructuration qu'il serait pertinent d'appliquer afin d'améliorer le code. Pour vous aider, rapportez-vous aux opérations vues en cours. N'indiquez pas le code restructuré.

a) Extrait de code C++ :

```
1  int main(void)
2  {
3    string numero_telephone="";
4    cout << "Svp., entrez votre numero de telephone:" << endl;
5    getline(cin, numero_telephone);
6
7    string code_zone="";
8    cout << "Svp., entrez votre nouveau code de zone:" << endl;
9    getline(cin, code_zone);
10
11   numero_telephone.replace(3, 3, code_zone);
12   cout << "Voici votre nouveau numero de telephone: " << numero_telephone << endl;
13 }
```

Problème rencontré (0,5 point) :
Manque d'abstraction.

Opération de restructuration à appliquer (1 point) :
Créer une classe NumeroTelephone avec des attributs pour les différentes parties d'un numéro plus un setter pour le code d'un zone.

6. Le problème qui vous apparaît le plus évident.

b) Extrait de code C++ :

```
1  class Employee
2  {
3   public:
4      int payAmount() {
5          switch (getType()) {
6              case EmployeeType.ENGINEER:
7                 return _monthlySalary;
8              case EmployeeType.SALESMAN:
9                 return _monthlySalary + _commission;
10             case EmployeeType.MANAGER:
11                return _monthlySalary + _bonus;
12             default:
13                throw RuntimeException("Incorrect Employee");
14         }
15     }
16     int getType() {
17         return _type.getTypeCode();
18     }
19  private:
20     EmployeeType _type;
21 }
22
23 class EmployeeType
24 {
25  public:
26     virtual int getTypeCode()=0;
27 }
28
29 class Engineer: public EmployeeType
30 {
31  public:
32     virtual int getTypeCode() {
33         return Employee.ENGINEER;
34     }
35 }
36
37 //similaire pour les autres sousclasses
38 //adapté de: \texttt{http://goo.gl/qs7WZ}
```

Problème rencontré (0,5 point) :
La classe EmployeeType possède un code de type qui est controlé, tandis qu'un switch-case est utilisé par le client pour obtenir la version concrète de l'implémentation.

Opération de restructuration à appliquer (1 point) :
Remplacer l'énoncé switch/case par une structure polymorphique. Introduire des sous-classes pour chacun des EmployeeTypes, mouvoir la méthode payAmount dans EmployeeType et l'implémenter différemment dans chaque sousclasse.

c) Extrait de code C++ :

```
1  int f(int a, Number** b){
2    int r = OK;
3    r = mem_alloc(10, (Number**) b);
4
5    if(r != OK){
6      /* no logging needed */
7      /* no deallocation needed */
8      return r;
9    }else{
10     if((a < 0)||(a > 10)){
11       r = PARAM_ERROR;
12       LOG(r,OK);
13       if(r != OK) mem_free(b);
14       return r;
15     }else{
16       r = g(a);
17       if(r != OK){
18         LOG(LINKED_ERROR,r);
19         r = LINKED_ERROR;
20         if(r != OK) mem_free(b);
21         return r;
22       }else{
23         r = h(b);
24         if(r != OK){
25           /* no logging needed */
26           if(r != OK) mem_free(b);
27           return r;
28         }else{
29           /* no deallocation needed */
30           return r;
31         }
32       }
33     }
34   }
35 }
```

Problème rencontré (0,5 point) :
Valeur d'erreur retournée, ce qui rend le code très compliqué.

Opération de restructuration à appliquer (1 point) :
Utilisation des exceptions.

d) Extrait de code C (notez que ceci n'est PAS du code orienté objet) :

```
1  static int EvaluatePatternExpression(
2    void *theEnv,
3    struct factPatternNode *patternPtr,
4    struct expr *theTest,
5    int thePosition)
6    {
7     DATA_OBJECT theResult;
8     struct expr *oldArgument;
9     int rv;
10
11    ...
12
13    /*=====================================*/
14    /* Evaluate pattern network primitives. */
15    /*=====================================*/
16
17    switch(theTest->type)
18      {
19       /*=============================================*/
20       /* This primitive compares the value stored in  */
21       /* a single field slot to a constant for either */
22       /* equality or inequality.                      */
23       /*=============================================*/
24
25       case FACT_PN_CONSTANT1:
26         oldArgument = EvaluationData(theEnv)->CurrentExpression;
27         EvaluationData(theEnv)->CurrentExpression = theTest;
28         rv = FactPNConstant1(theEnv,theTest->value,&theResult);
29         EvaluationData(theEnv)->CurrentExpression = oldArgument;
30         return(rv);
31
32       /*=============================================*/
33       /* This primitive compares the value stored in */
34       /* a multifield slot to a constant for either  */
35       /* equality or inequality.                     */
36       /*=============================================*/
37
38       case FACT_PN_CONSTANT2:
39         oldArgument = EvaluationData(theEnv)->CurrentExpression;
40         EvaluationData(theEnv)->CurrentExpression = theTest;
41         rv = FactPNConstant2(theEnv,theTest->value,&theResult);
42         EvaluationData(theEnv)->CurrentExpression = oldArgument;
43         return(rv);
44
45       /*=============================================*/
46       /* This primitive determines if a multifield slot */
47       /* contains at least a certain number of fields.  */
48       /*=============================================*/
49
50       case FACT_SLOT_LENGTH:
51         oldArgument = EvaluationData(theEnv)->CurrentExpression;
52         EvaluationData(theEnv)->CurrentExpression = theTest;
53         rv = FactSlotLength(theEnv,theTest->value,&theResult);
54         EvaluationData(theEnv)->CurrentExpression = oldArgument;
55         return(rv);
56
57       default:
58         print(``Continue function as normal\n'');
59      }
60
61    ...
62
63    return(TRUE);
64 }//adapté de: \texttt{http://clipsrules.sourceforge.net, fichier factmch.c.}
```

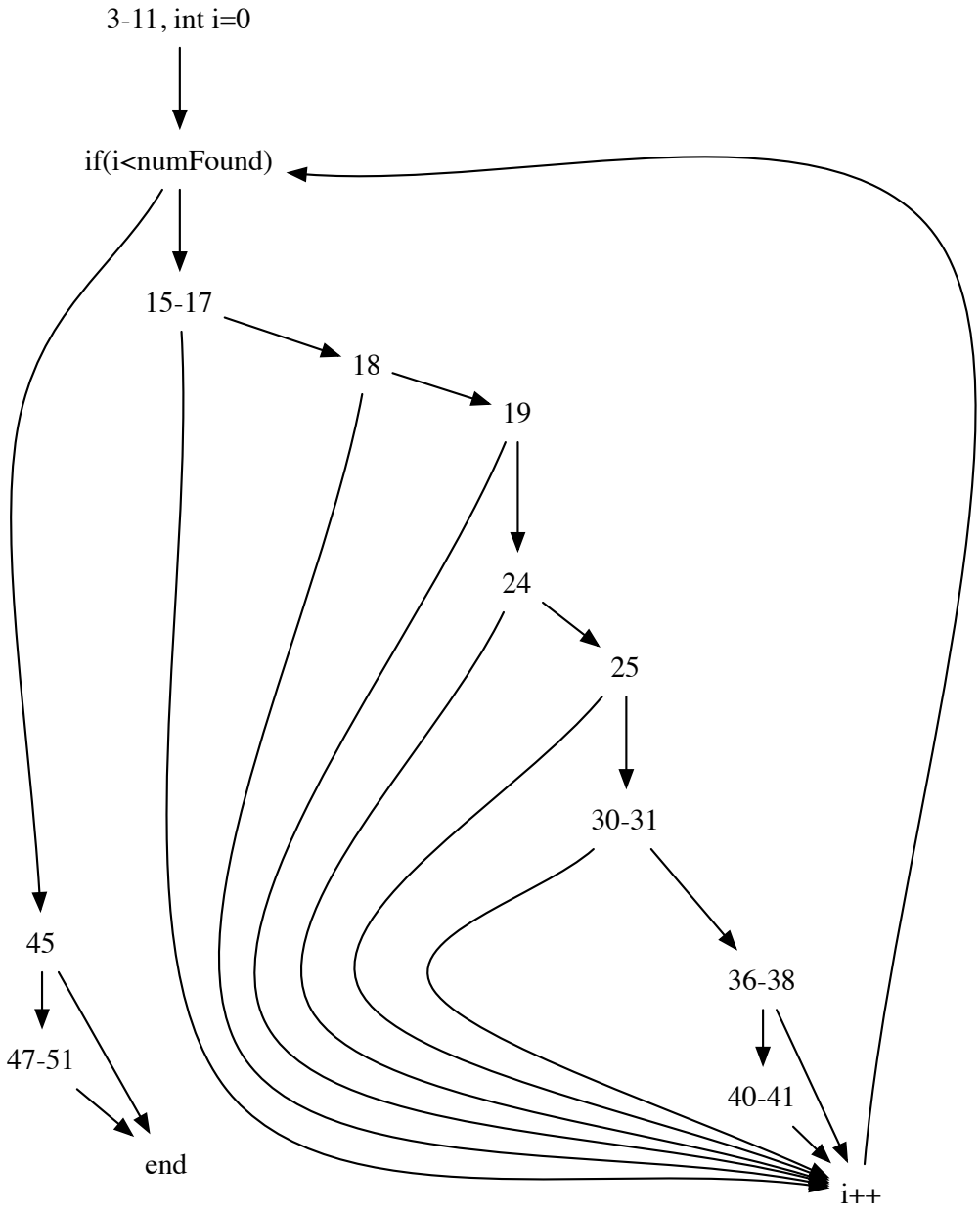
Problème rencontré (0,5 point) :
Duplication dans les différents case.

Opération de restructuration à appliquer (1 point) :
Extraire le code avant la ligne “rv = ...” et le code après du switch-case dans un macro.
Demi-point pour proposer des fonctions appelées avant et après.

8 Tests du développeur (7 points)

Pour la question qui suit, considérez à nouveau le code fourni **en annexe A**.

a) Tracez le graphe de flot de contrôle de la fonction *Seeker_FindEnemy*. Ne recopiez pas le code au complet dans les noeuds, n'utilisez que le ou les numéros de lignes, sauf où une ligne appartient à plusieurs nœuds (exemple : (1), (7,8,9), (a<45), etc.). (3 points) Version avec "continue" nodes also accepted, as well as version where "i++" or "continue" nodes are shared :



b) Quelle est la complexité cyclomatique de la fonction *Seeker_FindEnemy* et que signifie cette valeur dans le contexte de tests de couverture de code ? (2 points)
E=22, N=14, alors complexité cyclomatique : 22-14+2=10. Cela représente le nombre maximal de cas de tests nécessaires pour obtenir une couverture complète du code.

c) Donnez un ensemble minimal de tests nécessaires pour couvrir la fonction *Seeker_FindEnemy* entièrement, c.-à-d. :

(a) Donnez le nombre minimal de différents *entityList* avec lesquels il faut exécuter la fonction. (0.5 point)

(b) Donnez pour chaque tel *entityList* les caractéristiques de ses éléments, par exemple pour un *entityList* avec 2 éléments (dans cet exemple, le deuxième élément est arbitraire) : [ent->client->playerTeam == TEAM_NEUTRAL, arbitraire]. (1.5 point)

Au minimum 1 cas de test avec un *entityList* avec 1 élément satisfaisant à toutes les conditions suivantes (réponse correcte est 1 test, mais solutions avec plusieurs éléments acceptables aussi) :
- ent− >s.number != NPC− >s.number
- ent− >NPC
- ent− >health > 0
- ent− >client− >playerTeam != NPC− >client− >playerTeam
- ent− >client− >playerTeam ! = TEAM_NEUTRAL
- NPC_ClearLOS(ent)
- dis<=bestDis

Nous vous souhaitons beaucoup de succès dans le restant de vos études et / ou de votre cheminement de carrière !

Bram et Julien, enseignants du cours LOG1000

Annexe A

```
1  void Seeker_FindEnemy( void )
2  {
3   int   numFound;
4   float  dis, bestDis = SEEKER_SEEK_RADIUS * SEEKER_SEEK_RADIUS + 1;
5   vec3_t  mins, maxs;
6   gentity_t *entityList[MAX_GENTITIES], *ent, *best = NULL;
7
8   VectorSet( maxs, SEEKER_SEEK_RADIUS, SEEKER_SEEK_RADIUS, SEEKER_SEEK_RADIUS );
9   VectorScale( maxs, -1, mins );
10
11  numFound = gi.EntitiesInBox( mins, maxs, entityList, MAX_GENTITIES );
12
13  for ( int i = 0 ; i < numFound ; i++ )
14  {
15   ent = entityList[i];
16
17   if ( ent->s.number == NPC->s.number ||
18        !ent->NPC ||
19        ent->health <= 0 )
20   {
21    continue;
22   }
23
24   if ( ent->client->playerTeam == NPC->client->playerTeam ||
25        ent->client->playerTeam == TEAM_NEUTRAL ) // don't attack same team or bots
26   {
27    continue;
28   }
29
30   // try to find the closest visible one
31   if ( !NPC_ClearLOS( ent ))
32   {
33    continue;
34   }
35
36   dis = DistanceHorizontalSquared( NPC->currentOrigin, ent->currentOrigin );
37
38   if ( dis <= bestDis )
39   {
40    bestDis = dis;
41    best = ent;
42   }
43  }
44
45  if ( best )
46  {
47   // used to offset seekers around a circle so they don't occupy the same spot.
48   // This is not a fool-proof method.
49   NPC->random = random() * 6.3f; // roughly 2pi
50
51   NPC->enemy = best;
52  }
53 }
```

(adapté de : `git://git.code.sf.net/p/jedioutcast/code, fichier code/game/AI_Seeker.cpp`)

Annexe B

Implémentation de la classe `list` :

```
1  #include "List.h"
2  #include <cstdlib>
3  #include <iostream>
4  #include <stdexcept>
5
6  using namespace std;
7
8
9  List::List()
10   : numY_(0)
11 {
12   clean();
13 }
14
15 void List::insert(int newY)
16 {
17
18   if (numY_ = 0)  { // Y empty so far (easy case)
19     Y_[0] = newY;
20     return;
21   }
22   // need to insert just before the first Y
23   // element that newY is less than
24   for (int j = 0; j < numY_; j++)  {
25     if (newY < Y_[j])  {
26       // shift Y_[j], Y_[j+1],... rightward
27       // before inserting NewY
28       scootOver(j);
29       Y_[j] = newY;
30       return;
31     }
32   }
33 }
34
35 void List::scootOver(int jj)
36 {
37   int k;
38   for (k = numY_-1; k > jj; k++)
39     Y_[k] = Y_[k-1];
40 }
41
42
43 void List::getArgs(int ac, char **av)
44
45 {
46   int i;
47
48   numInputs_ = ac - 1;
49   for (i = 0; i < numInputs_; i++)
50     X_[i] = atoi(av[i+1]);
51 }
52
53
54 void List::processData()
55 {
56    for (numY_ = 0; numY_ < numInputs_; ++numY_){
57      // insert new Y in the proper place
58      // in list
59      insert(X_[numY_]);
60    }
```

```
61  }
62
63  void List::printResults()
64  {
65    int i;
66    for (i = 0; i < numInputs_; ++i)
67      cout << Y_[i] << endl;
68  }
69
70  void List::clean()
71  {
72    for (int i = 0; i < 10; ++i){
73      X_[i] = 0;
74      Y_[i] = 0;
75    }
76  }
77
78  // Pour les test
79  int List::getElement(int index)
80  {
81    if (index >= numY_)
82      throw out_of_range("Out of range");
83    return Y_[index];
84  }
```

Annexe C

Implémentation de la classe `ListTest` :

```
1  #include <cppunit/CompilerOutputter.h>
2  #include <cppunit/extensions/TestFactoryRegistry.h>
3  #include <cppunit/TestResult.h>
4  #include <cppunit/TestResultCollector.h>
5  #include <cppunit/TestRunner.h>
6  #include <cppunit/BriefTestProgressListener.h>
7
8  #include "ListTest.h"
9  #include <iostream>
10 #include <stdexcept>
11
12 using namespace std;
13
14 CPPUNIT_TEST_SUITE_REGISTRATION( ListTest );
15
16 void ListTest::setUp()
17 {
18   liste.clean();
19 }
20
21 void ListTest::tearDown()
22 {
23 }
24
25 // La liste d'item est vide
26 void ListTest::test1()
27 {
28   CPPUNIT_ASSERT_THROW(liste.getElement(0),out_of_range);
29 }
30
31 // On insère un seul élément
32 void ListTest::test2()
33 {
34   liste.insert(5);
35   CPPUNIT_ASSERT_EQUAL(5,liste.getElement(0));
36 }
37
38 // On insère deux éléments en ordre
39 void ListTest::test3()
40 {
41   liste.insert(5);
42   liste.insert(10);
43   CPPUNIT_ASSERT_EQUAL(5,liste.getElement(0));
44   CPPUNIT_ASSERT_EQUAL(10,liste.getElement(1));
45 }
46
47 // On insère deux éléments en désordre
48 void ListTest::test4()
49 {
50   liste.insert(10);
51   liste.insert(5);
52   CPPUNIT_ASSERT_EQUAL(5,liste.getElement(0));
53   CPPUNIT_ASSERT_EQUAL(10,liste.getElement(1));
54 }
55
56 // On insère trois fois le même élément
57 void ListTest::test5()
58 {
59
60   liste.insert(5);
```

```
61    liste.insert(5);
62    liste.insert(5);
63    CPPUNIT_ASSERT_EQUAL(5,liste.getElement(0));
64    CPPUNIT_ASSERT_EQUAL(5,liste.getElement(1));
65    CPPUNIT_ASSERT_EQUAL(5,liste.getElement(2));
66
67  }
68
69  // Cas général
70  void ListTest::test6()
71  {
72
73    liste.insert(5);
74    liste.insert(2);
75    liste.insert(57);
76    liste.insert(12);
77    liste.insert(9);
78    liste.insert(18);
79    liste.insert(45);
80    liste.insert(3);
81    liste.insert(1);
82    liste.insert(100);
83    CPPUNIT_ASSERT_EQUAL(1,liste.getElement(0));
84    CPPUNIT_ASSERT_EQUAL(2,liste.getElement(1));
85    CPPUNIT_ASSERT_EQUAL(3,liste.getElement(2));
86    CPPUNIT_ASSERT_EQUAL(5,liste.getElement(3));
87    CPPUNIT_ASSERT_EQUAL(9,liste.getElement(4));
88    CPPUNIT_ASSERT_EQUAL(12,liste.getElement(5));
89    CPPUNIT_ASSERT_EQUAL(18,liste.getElement(6));
90    CPPUNIT_ASSERT_EQUAL(45,liste.getElement(7));
91    CPPUNIT_ASSERT_EQUAL(57,liste.getElement(8));
92    CPPUNIT_ASSERT_EQUAL(100,liste.getElement(9));
93  }
94
95  // Les deux premiers en ordre, le troisième doit être inséré au début
96  // de la liste
97  void ListTest::test7()
98  {
99    liste.insert(2);
100   liste.insert(3);
101   liste.insert(1);
102   CPPUNIT_ASSERT_EQUAL(1,liste.getElement(0));
103   CPPUNIT_ASSERT_EQUAL(2,liste.getElement(1));
104   CPPUNIT_ASSERT_EQUAL(3,liste.getElement(2));
105 }
```