CPP examen 12/12/2016

Question 1

```
void Complete(std::vector<int>& v1, std::vector<int>& v2) {
        while(v1.size()!=v2.size()) (v1.size()>v2.size())?v2.push_back(0):v1.push_back(0);
}

std::vector<int> vector1;
std::vector<int> vector2;
for(int i=0; i<3; i++) vector1.push_back(0);
for(int i=0; i<12; i++) vector2.push_back(0);
Complete(vector1, vector2);
std::cout << vector1.size() == vector2.size() << std::endl;
```

Question 2

f1 est une fonction membre de la classe Strange.
f1 prend un objet B en paramètre et renvoi une référence d'objet A.
aB est une copie de l'objet B passé en paramètre à f1 et sera donc détruit à la fin du bloc.

Question 3

Une classe abstraite permet de faire du polymorphisme car elle peut être étendue par d'autres classes en étant ainsi une classe mère.
Elle ne peut pas être instanciée.
Une classe abstraite possède au moins une fonction membre virtuelle pure.
Les destructeurs d'une classe abstraite doivent être abstraits.
Il faut définir une fonction clone virtuelle appelé par le constructeur de copie et l'opérateur d'affectation pour conserver le type dynamique.

Question 4

- appel de f1 membre de C1
- appel de f1 membre de C2
- appel de f1 membre de C1
- appel de f1 membre de C1
- appel de f1 membre de C1
- appel de f1 membre de C1

- appel de f2 membre de C1
- appel de f2 membre de C1
- appel de f2 membre de C3
- appel de f2 membre de C1
- appel de f2 membre de C1
- appel de f2 membre de C3

Question 5

1)

internalA est un attribut de A de type « public A* »
Pas de référence car 0..1 implique qu'il peut être nul

Pas l'objet car allocation infinie de mémoire possible sinon (et internalA pourrait être de type Aplusplus, on évite aussi la troncature)

2)

```
void setInternalA(A* a);
void A::setInternalA(A* a) {
	delete(internalA);
	internalA = a.clone();
}

virtual ~A();
A::~A() {
	delete(internalA);
	internalA = nullptr;
}

virtual ~Aplusplus();
Aplusplus::~Aplusplus() {
	delete(&anAttribute);
	anAttribute = nullptr;
}

virtual A* colne();
A* A::clone() {
	return new A(*this);
}

virtual A* clone();
A* Aplusplus::clone() {
	return new Aplusplus(*this);
}

A(const A& a);
A::A(const A& a) {
	if(a.internalA!= nullptr) internalA = (a.internalA)->clone();
}

Aplusplus(const Aplusplus& a);
Aplusplus::Aplusplus(const Aplusplus& a) {
	anAttribute = a.anAttribute ;
}
```

Question 6

A)

Le return de la fonction get_i_plus_j() n'est pas bon, 'i' n'existe pas, et l'attribut 'pti' de A est un pointeur, il faudrait donc le déréférencer.
De plus, le constructeur pas défaut de B est appelé dans la fonction main, or ce constructeur n'existe pas. Et il est possible que ce constructeur par défaut appelle celui de la classe A qu'il faudrait alors créer.

```
int B::get_i_plus_j() {
	return (pti==nullptr)?j:(*pti + j);
}

class B : public A {
public :
	B(int i = 0): j(i){}
…
}

class A : {
public :
	A(int i = 0): pti(new int(i)){}
…
}
```

B)

Tout d'abord, il faut penser à mettre le destructeur de A en virtuel car il peut être étendu par d'autres classes en ayant besoin.

Ensuite il faut définir un constructeur par copie car, si une copie de A est faite, la copie par défaut va recopier le pointeur bit à bit et les deux objets partageront la même valeur pointée.

Question 7

A(char c); OU A(int i);
A(std::string s, void* v);
C() = default;
virtual A(const A& a); ET C doit hériter de A donc on ajoute un constructeur par défaut à A
friend C& operator*(const double d, const C& c);

Question 8

Impossible… OSKOUR !