Exercice 1

1. On a l'arbre de preuve :

$$\dfrac{\dfrac{\dfrac{}{\neg A, A \vdash \neg A}\,\text{ax} \qquad \dfrac{}{\neg A, A \vdash A}\,\text{ax}}{\neg A, A \vdash \bot}\,\neg_e}{\vdash \neg A \to (A \to \bot)}\,\to_i \times 2$$

2. On a l'arbre de preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \to \bot, A \vdash A}\,\text{ax} \qquad \dfrac{}{A \to \bot, A \vdash A \to \bot}\,\text{ax}}{A \to \bot, A \vdash \bot}\,\to_e}{A \to \bot \vdash \neg A}\,\neg i}{\vdash (A \to \bot) \to \neg A}\,\to_i$$

3. Introduction : $\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_i$; élimination : $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge_e^g$ et $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge_e^d$. On a alors la preuve (pas très intéressante, mais bon) : $\dfrac{\dfrac{\text{Question 1}}{\vdash \neg A \to (A \to \bot)} \qquad \dfrac{\text{Question 2}}{\vdash (A \to \bot) \to \neg A}}{(\neg A \to (A \to \bot)) \wedge ((A \to \bot) \to \neg A)}\,\wedge_i$

4. Dressons la table de vérité de $((x \to y) \to x) \to x$:

x	y	$x \to y$	$(x \to y) \to x$	$((x \to y) \to x) \to x$
0	0	1	0	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

   Comme il s'agit d'une tautologie et par le théorème de substitution (non attendu à mon avis pendant l'oral), on en déduit que P est une tautologie.

5. On complète :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\,\text{ax} \qquad \dfrac{}{\Gamma, A \vdash \neg A}\,\text{ax}}{\Gamma, A \vdash \bot}\,\neg_e}{\Gamma, A \vdash B}\,\bot_e}{\Gamma \vdash A \to B}\,\to_i \qquad \dfrac{}{\Gamma \vdash (A \to B) \to A}\,\text{ax}}{\Gamma \vdash A}\,\to_e$$

Exercice 2

1. On propose d'abord un calcul de factorielle :

```
uint64_t fact(int n){
    uint64_t f = 1;
    for (int i=2; i<=n; i++){
        f *= i;
    }
    return f;
}
```

Ensuite, on peut compléter le code :

```
uint64_t catalan(int n) {
    return fact(2 * n) / (fact(n + 1) * fact(n));
}
```

2. On va avoir assez rapidement un dépassement d'entiers pour le calcul de factorielle, ce qui rendra faux le résultat d'un appel à `catalan`. On constate effectivement ce problème à partir de $n = 11$.
3. On garde en mémoire un compteur qui indique la différence entre le nombre de parenthèses ouvrantes et fermantes au cours de la lecture du mot. On vérifie que ce compteur reste toujours positif, et qu'il est nul à la fin du mot.

```
bool verification(char * mot) {
    int diff = 0;
    for (int i=0; mot[i] != '\0'; i++){
        if (mot[i] == '(') diff++;
        else {
            diff--;
            if (diff <0) return false;
        }
    }
    return (diff == 0);
}
```

4. La complexité est linéaire en la taille du mot.
5. On doit générer toutes les chaînes de taille $2n$ sur un alphabet de taille 2, puis faire la vérification sur chacune de ces chaînes. La complexité totale serait donc en $\mathcal{O}(n \times 2^{2n}) = \mathcal{O}(n \times 4^n)$.
6. On suit l'algorithme qui nous est décrit. On pense à ajouter un caractère de fin de chaîne pour l'affichage.

```
void dyck(char s[N], int o, int f, int n) {
    if (o == f && f == n) {
        s[2 * n] = '\0';
        printf("%s\n", s);
        return;
    }
    if (o < n){
        s[o + f] = '(';
        dyck(s, o + 1, f, n);
    }
    if (f < o){
        s[o + f] = ')';
        dyck(s, o, f + 1, n);
    }
}
```

7. On s'inspire du fait que le code contient une variable `nbmots`, qu'on rajoute en argument à la fonction (sous forme de pointeur pour pouvoir la modifier).

```c
void dyck(char s[N], int o, int f, int n, uint64_t* nbmots) {
    if (o == f && f == n) {
        (*nbmots)++;
        return;
    }
    if (o < n){
        s[o + f] = '(';
        dyck(s, o + 1, f, n, nbmots);
    }
    if (f < o){
        s[o + f] = ')';
        dyck(s, o, f + 1, n, nbmots);
    }
}
```

On trouve 35357670 mots de Dyck pour $n = 16$.

8. Cette question n'est pas difficile mais est un peu technique et n'est là a priori que pour occuper les candidats les plus rapides.