NOM : Prénom :

Examen Méthodologie Orientée Objets / C++ (IF2)

Durée 1h15, documents autorisés, annales et appareils communiquant interdits

→Répondre sur le sujet←

Questions fondamentales (max 10 minutes)

1) Quelle est la taille maximale d'un tableau de *char* ? :

2) Parmi les 4 variables a, b, c et d suivantes, entourer celle(s) occupant la plus grande place mémoire ? char *a ; short *b ; int *c ; double *d ;
Justifier:

3) Si on a : char t[]=''hello'' ; donner ce qui sera affiché par l'instruction :
cout << (int) t[5] ;

4) Que vaut la variable x telle que : double x=1/2 ; ? :

5) Ecrire la fonction *Swap_Me* qui échange le contenu des variables. On donne son utilisation dans un main :

```
        double pi = 3.14159265 ;
        double e = 2.718281;
        Swap_Me(&pi, &e) ;
// maintenant e vaut 3.14159265 et pi 2.718281
```

Exercice analyse (15 minutes)

Voici le code à analyser où on initialise 4 variables constantes avec des chaines de caractères identiques :

```
int main()
{
char *s1 = "Hello From GE!";
char *s2 = "Hello From GE!";

char s3[] = "Hello From GE!";
char s4[] = "Hello From GE!";

cout<< "s1 : "<< s1 << " "<<hex<< &s1 <<" "<<(int *) &(s1[0])<<" "<<dec<< sizeof(s1)<< endl;
cout<< "s2 : "<< s2 << " "<<hex<< &s2 <<" "<<(int *) &(s2[0])<<" "<<dec<< sizeof(s2)<< endl;
cout<< "s3 : "<< s3 << " "<<hex<< &s3 <<" "<<(int *) &(s3[0])<<" "<<dec<< sizeof(s3)<< endl;
cout<< "s4 : "<< s4 << " "<<hex<< &s4 <<" "<<(int *) &(s4[0])<<" "<<dec<< sizeof(s4)<< endl;

    return 0;
}
```

Une fois compilé puis exécuté, voici ce qui est affiché par le programme :

```
s1 : Hello From GE! 0x22fe38 0x405000 8
s2 : Hello From GE! 0x22fe30 0x405000 8
s3 : Hello From GE! 0x22fe20 0x22fe20 15
s4 : Hello From GE! 0x22fe10 0x22fe10 15
```

6) Donner la signification des valeurs numériques affichées.

7) Justifier pour les lignes correspondant à s3 et s4 la cohérence des valeurs.

8) Expliquer ce qui a été fait par le compilateur pour les variables s1 et s2.

9) En déduire les différences entre les déclarations sur constante de type char *s et char s[].

10) Donner le code permettant d'allouer (et libérer) dynamiquement un tableau de double *t*, défini ainsi : *double *t ;*

Exercice : Diagramme d'états (45 minutes)

Le but de cet exercice est de concevoir et programmer des classes permettant de modéliser des diagrammes d'états puis de les exécuter.

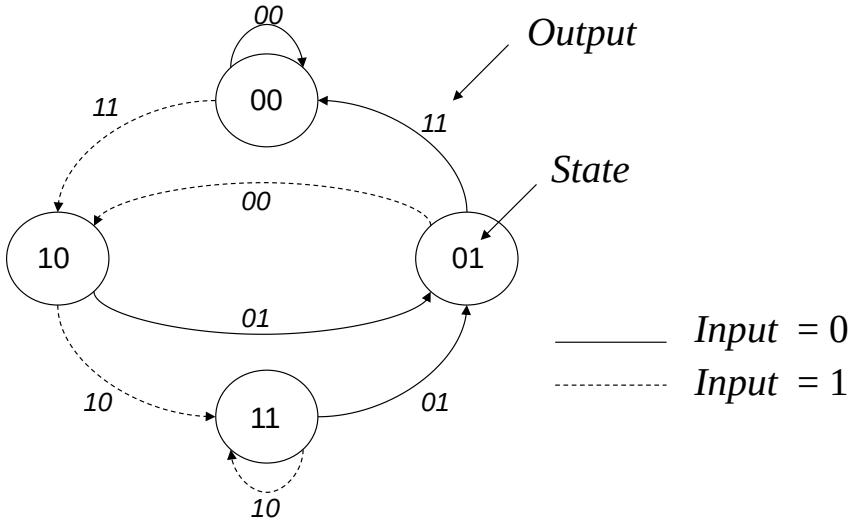
Une machine d'états (ou automate fini) est constituée d'un ensemble d'états et d'un ensemble de transitions. Les transitions relient un état d'origine vers un état d'arrivé.

Un diagramme d'états représente les états (***State***), les transitions (***Transition***) et les comportements d'une machine d'états.

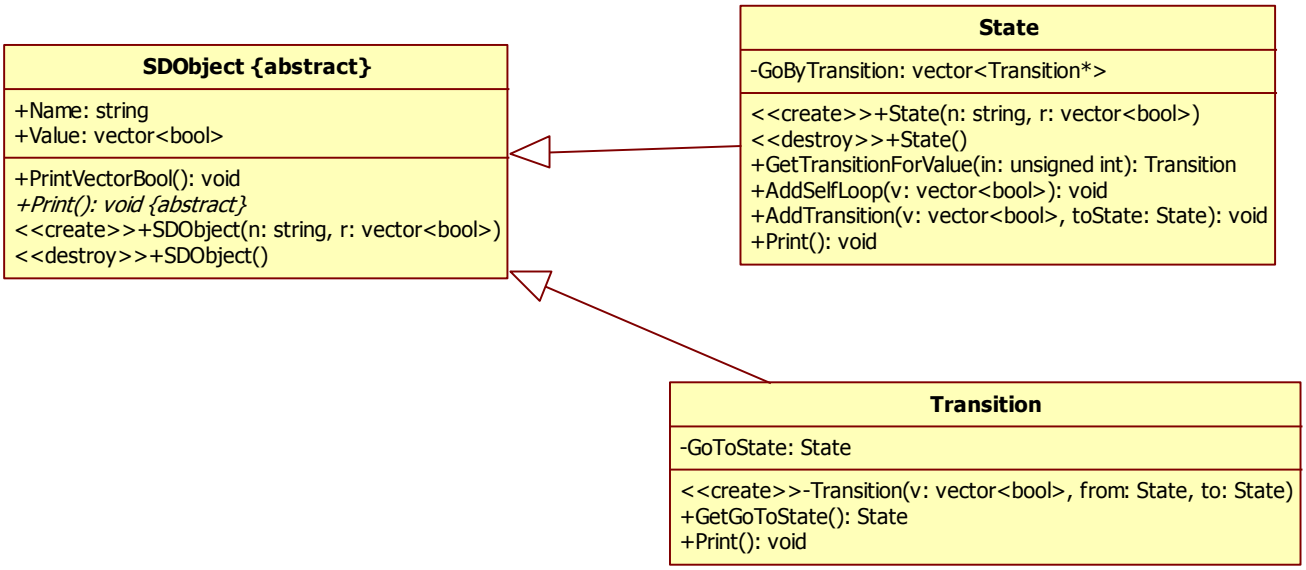
Une machine d'états passe d'un état à un autre, via une transition, en fonction des valeurs d'entrées (***Input***). Lors d'une transition, des valeurs de sortie (***Output***) sont produites.

Dans l'exemple suivant, un diagramme d'états à 4 états est représenté, avec ses transitions et les deux bits produits lors d'une transition. Dans cet exemple, l'entrée n'est que d'un bit.

Ainsi, si on se trouve à l'état « 01 » et qu'un 0 arrive en entrée, alors on passe à l'état « 00 » en produisant en sortie « 11 ».



Classes SDObjects, Transition et State (20 minutes)



Le diagramme précédent est proposé pour modéliser les classes fondamentales du système. On remarquera les notations *vector< bool>* et *vector < Transition *>*.

On donne aussi le code du constructeur de ***Transition*** :

```
Transition::Transition(const vector<bool> &v, const State* from, State *to):
SDObject(from->Name + "->" + to->Name, v)
{   GoToState = to; }
```

et le code la fonction *AddTransition* de la classe **State** qui permet l'ajout d'une transition à un état :

```
void State::AddTransition(const vector<bool> &v, State *toState)
{
    Transition *t = new Transition(v, this, toState);
    GoByTransition.push_back(t);
}
```

A noter :

- le champ *Name* permet de donner un nom à une transition ou à un état.
- Le champ *Value* permet de coder une valeur binaire sous forme d'un tableau de booléens. Celle-ci correspondra soit à la valeur d'un état, soit à la valeur à produire en sortie pour une transition.

1) Donner la définition C++ de la classe ***SDObject. (sur le diagramme UML précédent)***
2) Quelles méthodes doivent être surchargées par les classes filles de ***SDObject*** ?

Voici les définitions C++ des classes ***Transition*** et ***State*** :

```
class Transition: public SDObject
{private:
    State *GoToState;
public:
    Transition(const vector<bool> &v, const State* from, State *to);
    State * GetGoToState() {return GoToState;}
    void Print() const;
};
```

```
class State: public SDObject
{ private:
    vector<Transition*> GoByTransition;
  public:
    State(const string &n, const vector<bool> &r);
    virtual ~State();
     Transition * GetTransitionForValue(unsigned int in) {return GoByTransition[in];}
     void AddSelfLoop  ( const vector<bool> &v );
     void AddTransition(const vector<bool> &v, State *toState);
     void Print() const;
};
```

3) Donner le code C++ du constructeur de la classe ***State***.

4) Justifier la présence d'un destructeur dans la classe ***State***.

5) Donner le code C++ de la fonction de *State::AddSelfLoop* qui permet d'ajouter une transition (et une valeur) à un état sur lui-même (sur l'exemple : états « 00 » et « 11 »).

6) Compléter le code suivant (~8 lignes) afin de reproduire le diagramme donné en exemple. Attention à l'ordre des transitions…

```
vector<State> S;
S.push_back( State ("A",{0,0}) );
S.push_back( State ("B",{1,0}) );
S.push_back( State ("C",{1,1}) );
S.push_back( State ("D",{0,1}) );
```

Classe StateDiagram (20 minutes)

On souhaite maintenant ajouter une classe ***StateDiagram*** pour le concept de diagramme d'états. Cette classe permettra de contenir un diagramme d'états et de le faire fonctionner par rapport à une entrée. Notamment, il faudra pouvoir :

- se souvenir de l'état courant (celui qui est actif)
- initialiser à l'état « 00 » le diagramme
- disposer de la méthode ***RunOnce*** qui permet de changer d'état à partir d'une valeur d'entrée (*input*) sous forme d'un tableau de booléens ou d'un entier. Cette fonction retourne la valeur de la transition.
- disposer de la fonction ***PrintRun*** qui à partir d'un tableau de *n* valeurs d'entrée, affiche successivement les *n* valeurs des transitions effectuées.

7) **Compléter le diagramme de classes avec celui de la classe *StateDiagram*.**
8) Justifier les relations entre ***StateDiagram*** et les trois autres classes.

9) Ecrire la fonction *StateDiagram ::RunOnce* :

Fin.