C++

Mot clé static
Exceptions

throw
catch

T::function();

Mot clé static

- **Variable static** = variable définie qu'une seule fois
  - la durée de vie de la variable est celle du programme
- **Fonction membre static** = fonction qui ne nécessite pas d'instance d'une classe pour s'éxecuter.

Variable static

```
void fonction()
{
    static int a=6;
    a++;
    std::cout<<a<<std::endl;
}

int main()
{
    fonction();
    fonction();
    fonction();
}
```

Variable définit
qu'une seule fois à 6

7
8
9

existe en C

Variable de classe static

```
struct objet
{
    objet() {nbr_occurence++;}
    static int nbr_occurence;
};

int objet::nbr_occurence=0;

int main()
{
    objet o1,o2,o3;
    objet o4;

    std::cout<<o1.nbr_occurence<<std::endl;
    std::cout<<objet::nbr_occurence<<std::endl;
}
```

Variable de classe static

Obligation de définir la variable une unique fois (dans un .cpp).

Appel depuis une instance

Appel depuis le nom de l'objet

```
4
4
```

Variable de classe static

```
struct objet
{
    objet() {}
    static const int constante=8;
};


int main()
{
    objet o1;

    std::cout<<o1.constante<<std::endl;
    std::cout<<objet::constante<<std::endl;
}
```

Initialisation possible pour des paramètres static constants

Appel depuis une instance

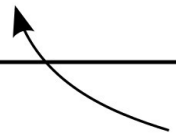
Appel depuis le nom de l'objet

```
8
8
```

Fonction membre static

```
struct math
{
    static float square(float x) {return x*x;}
    static float cube(float x) {return x*x*x;}
    static float sqrt(float x) {return std::sqrt(x);}
};


int main()
{
    std::cout<<math::square(1.5)<<std::endl;
}
```



Fonctions membres static
n'ont pas besoin d'une instance
de la classe.

Fonction membre static

```
class vec2
{
private:
    float x_data,y_data;
    static int alpha;

public:
    vec2(float x_arg,float y_arg):x_data(x_arg),y_data(y_arg){}
    float x() const {return x_data;}
    float y() const {return y_data;}

    static void set_alpha(int alpha_arg){alpha=alpha_arg;}

    static float norm(const vec2& v)
    {
        return std::pow(std::pow(v.x(),alpha)+std::pow(v.y(),alpha),1.0f/alpha);
    }
};

int vec2::alpha=2;
```

```
int main()
{
    vec2 v(1.2,3.3);

    std::cout<<vec2::norm(v)<<std::endl;
    vec2::set_alpha(4);
    std::cout<<vec2::norm(v)<<std::endl;
}
```

Fonction membre static

```
class vec2
{
private:
    float x_data,y_data;
    static int alpha;

public:
    vec2(float x_arg,float y_arg):x_data(x_arg),y_data(y_arg){}
    float x() const {return x_data;}
    float y() const {return y_data;}

    static void set_alpha(int alpha_arg){alpha=alpha_arg;}

    static float norm(const vec2& v)
    {
        return std::pow(std::pow(v.x(),alpha)+std::pow(v.y(),alpha),1.0f/alpha);
    }
};

int vec2::alpha=2;
```

Fonction membre static peut accéder aux attributs static

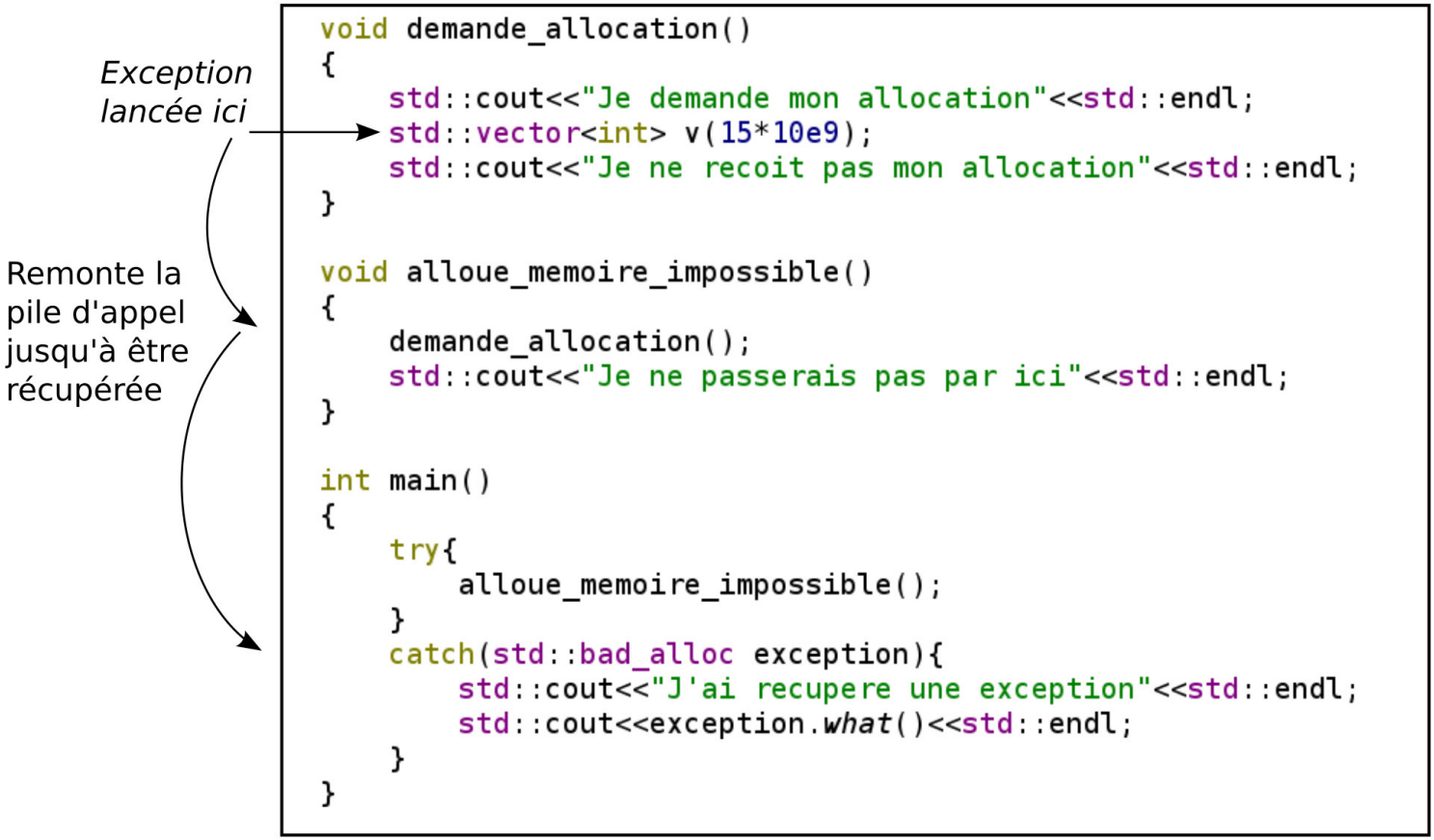
Initialisation (même si attribut privé)

Appel par nom_classe::fonction()

```
int main()
{
    vec2 v(1.2,3.3);

    std::cout<<vec2::norm(v)<<std::endl;
    vec2::set_alpha(4);
    std::cout<<vec2::norm(v)<<std::endl;
}
```

Exception




```
void demande_allocation()
{
    std::cout<<"Je demande mon allocation"<<std::endl;
    std::vector<int> v(15*10e9);
    std::cout<<"Je ne recoit pas mon allocation"<<std::endl;
}

void alloue_memoire_impossible()
{
    demande_allocation();
    std::cout<<"Je ne passerais pas par ici"<<std::endl;
}

int main()
{
    try{
        alloue_memoire_impossible();
    }
    catch(std::bad_alloc exception){
        std::cout<<"J'ai recupere une exception"<<std::endl;
        std::cout<<exception.what()<<std::endl;
    }
}
```

Je demande mon allocation
J'ai recupere une exception
std::bad_alloc

Exception

Une exception non récupérée **termine** le programme

```
int main()
{
    alloue_memoire_impossible();
}
```

Je demande mon allocation
terminate called after throwing an instance of 'std::bad_alloc'
what(): std::bad_alloc
The program has unexpectedly finished.

Exception

```
#include <vector>
#include <stdexcept>
struct vec2 {int x,y;};

int main()
{
    vec2* v1=new vec2;
    vec2* v2=new vec2;
    std::vector<vec2*> T={v1,v2};

    try
    {
        vec2 b=*T.at(3);
        std::cout<<b.x<<","<<b.y<<std::endl;
    }
    catch(std::out_of_range e)
    {
        std::cout<<"Je suis alle trop loin, je libere ma memoire"<<std::endl;
        std::cout<<"J'ai recupere "<<e.what()<<std::endl;
        for(unsigned int k=0;k<T.size();++k)
            delete T[k];
        std::cout<<"Memoire libere"<<std::endl;
    }
}
```

lance une exception

Si le pgm continuait, il faut libérer la mémoire allouée

Je suis alle trop loin, je libere ma memoire
J'ai recupere vector::_M_range_check
Memoire libere

Exception

```
#include <vector>
#include <stdexcept>
struct vec2 {int x,y;};

int main()
{
    vec2* v1=new vec2;
    vec2* v2=new vec2;
    std::vector<vec2*> T={v1,v2};

    try
    {
        vec2 b=*T.at(3);
        std::cout<<b.x<<","<<b.y<<std::endl;
    }
    catch( std::exception const & e)
    {
        std::cout<<"Je suis alle trop loin, je libere ma memoire"<<std::endl;
        std::cout<<"J'ai recupere "<<e.what()<<std::endl;
        for(unsigned int k=0;k<T.size();++k)
            delete T[k];
        std::cout<<"Memoire libere"<<std::endl;
    }
}
```

Si l'on ne connait pas l'exception: utilisation du polymorphisme

Je suis alle trop loin, je libere ma memoire
J'ai recupere vector::_M_range_check
Memoire libere

Lancer sa propre exception

```
void ecrit_fichier(const std::string filename)
{
    std::ofstream ofs(filename);
    if(ofs.good()==false)
        throw std::string("Fichier impossible a ouvrir");

    ofs<<"Bonjour"; ofs.close();
}

int main()
{
    std::string filename="dossier_inexistant/fichier_inexistant";
    try{ecrit_fichier(filename);}
    catch(  std::string const & s)
    {
        std::cout<<"Exception recuperee"<<std::endl;
        std::cout<<s<<std::endl;
    }
}
```

throw: lancer une exception

Lancer sa propre exception

```
class mon_exception
{
public:
    mon_exception(const std::string& fichier_arg,const std::string& type_erreur_arg)
        :fichier(fichier_arg),type_erreur(type_erreur_arg){}

    std::string info() const
    {
        return "Erreur pour le fichier "+fichier+" : "+type_erreur;
    }

private:
    std::string fichier;
    std::string type_erreur;
};
```

Classe d'exception

```
void ecrit_fichier(const std::string filename)
{
    std::ofstream ofs(filename);
    if(ofs.good()==false)
        throw mon_exception(filename,"Impossible a ouvrir");

    ofs<<"Bonjour"; ofs.close();
}
```

Lancement de l'exception

```
int main()
{
    std::string filename="dossier_inexistant/fichier_inexistant";
    try{ecrit_fichier(filename);}
    catch( mon_exception const & e)
    {
        std::cout<<"Exception recuperee"<<std::endl;
        std::cout<<e.info()<<std::endl;
    }
}
```

Gestion d'exception