C

Les références

T const&

000

Passage d'argument Rappel: Les paramètres sont passés par copies en C et en C++ #include <iostream> b est une copie de a int ma_fonction(int b) (même valeur, emplacement mémoire différent) b=b+2; ← return b; modification locale sur b n'affecte pas a int main() int a=5; int c=ma_fonction(a); c est une copie de b std::cout<<a<<","<<c<std::endl; return 0; 001

Passage d'argument Rappel: En C, pour modifier un paramètre dans une fonction, on passe son adresse. #include <iostream> void ma_fonction(int *b) b est une copie de pa b contient la copie de l'adresse de a *b=*b+2; donc b pointe vers a int main() on modifie la valeur pointée => on modifie a int a=5; int *pa=&a; ma_fonction(pa); std::cout<<a<<std::endl; return 0; 002

```
Pointeurs = problèmes
 Pointeurs vers valeurs non prévues

    Segmentation Fault

    Modification de variables imprévues

 Coder (proprement) avec des pointeurs demande beaucoup de contraintes:

    Faire attention à l'initialisation

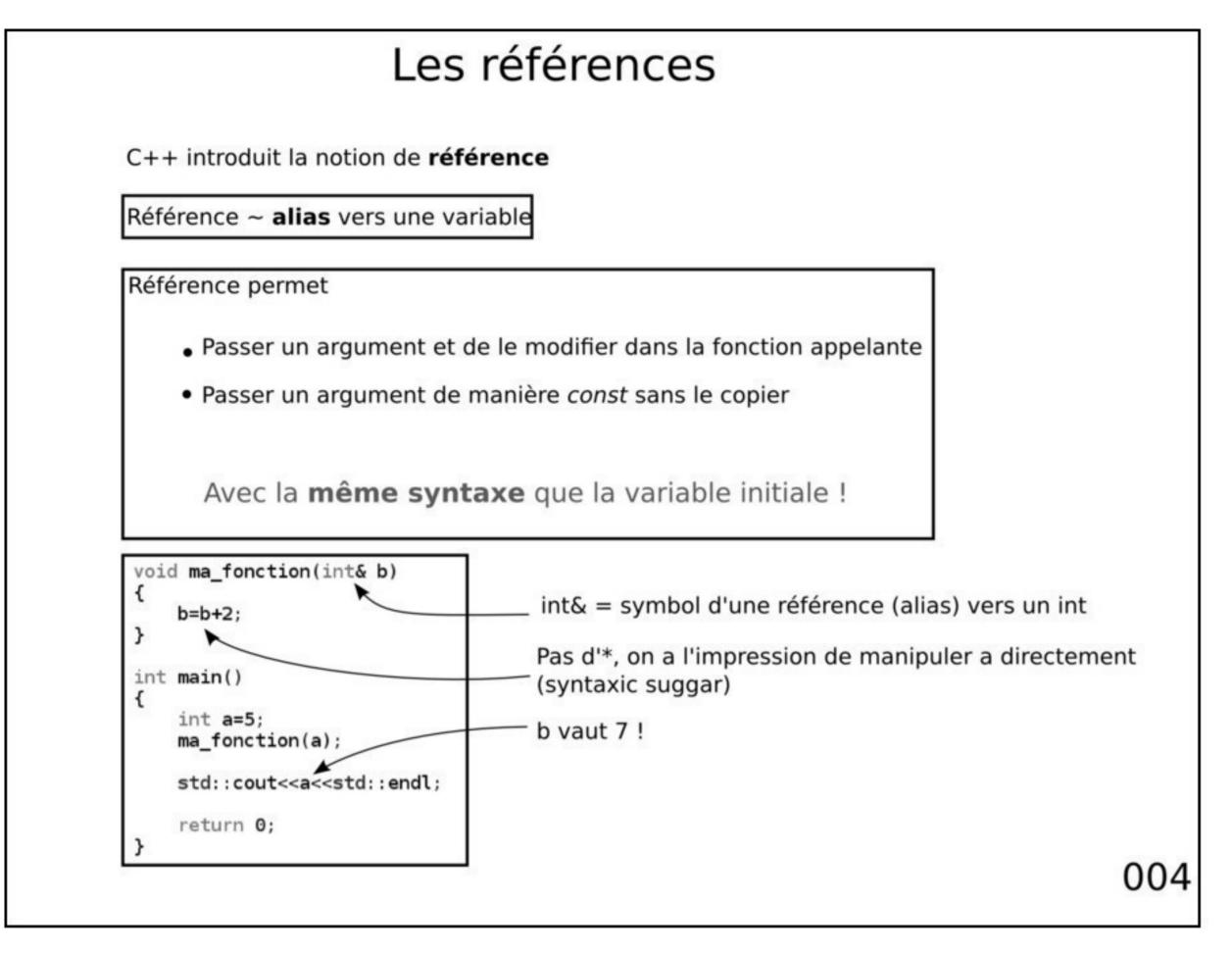
                                                  int *p=NULL:

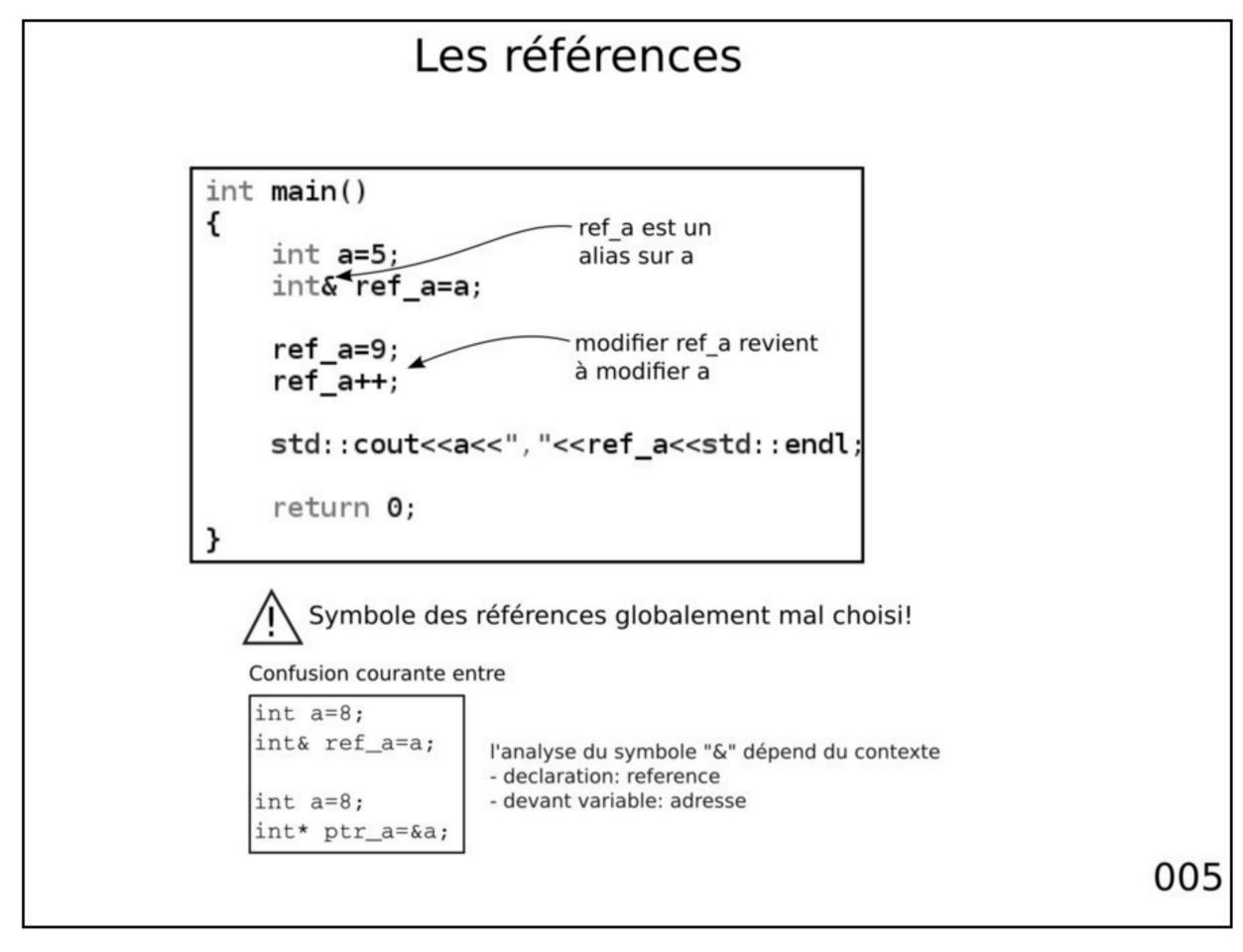
    Vérifier si pointeur non NULL

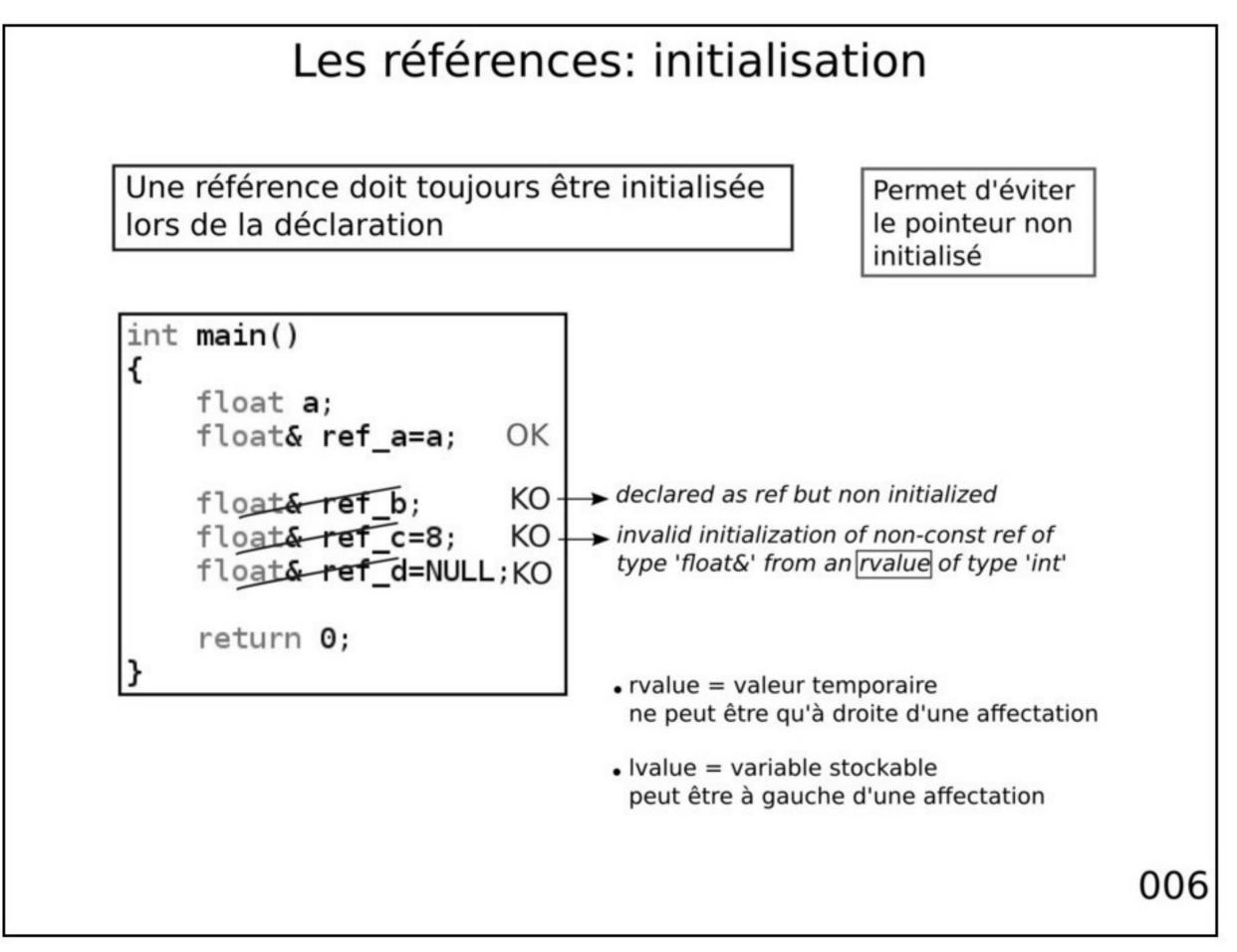
                                                  assert (p!=NULL);

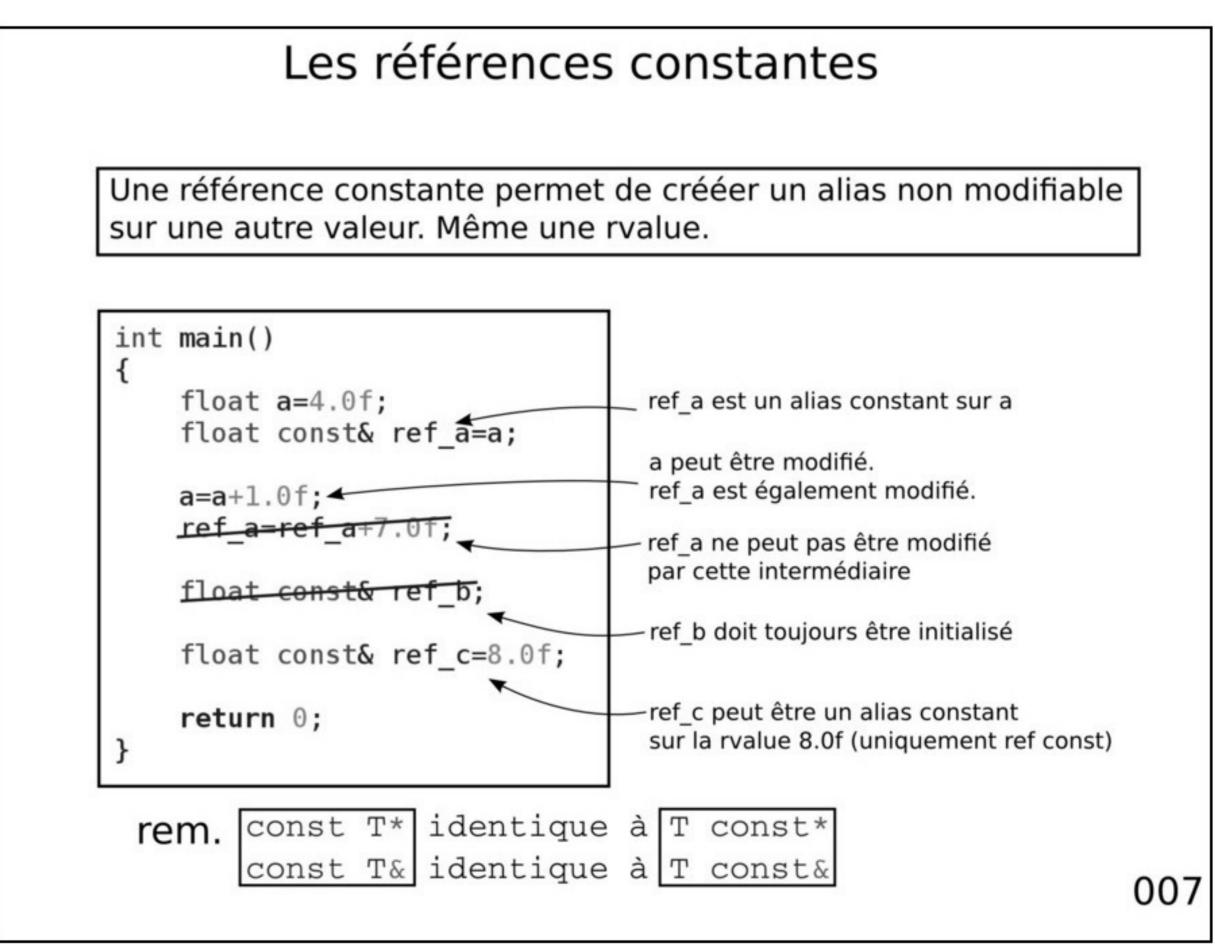
    Changement de la syntaxe du code

                                                  *p=*p+1;
             (beaucoup d'*)
Utilisation principale des pointeurs:
                                                                  f(int *p)
                                                                  { *p=*p+1; ...
     1. Passer un argument par son adresse pour le modifier
     2. Passer une struct par son adresse const sans la modifier
                                                                 f(int const* p)
                                                                  { int a=*p; ...
"Sous-utilisation" de la puissance des pointeurs
                                                                                      003
```









Intérêt des références

```
struct vec4
    double x,y,z,w;
                                   passage d'une référence
void multiply(vec4& v,double s)
    V. x *= s;
                           modification de la variable
    v.y *= s;
                           comme une variable locale
    v.z *= s;
                           pas d'*, pas de ->
    V.W *= S;
                           "sucre syntaxique"
int main()
    vec4 v={1.1,7.4,8.8,9.1};
    multiply(v, 4.0);
    std::cout<<v.x<<" "<<v.y<<" "<<v.z<<" "<<v.w<<std::endl;
    return 0;
```

Intérêt des références

```
struct vec4
{
    double x,y,z,w;
    pas de copie inutile (rapide)
    pas de syntaxe pointeur inutile

void print( vec4 const& v)
{
    std::cout<<v.x<<" "<<v.y<<" "<<v.z<<" "<<v.w<<std::endl;
}

int main()
{
    vec4 v={1.1,7.4,8.8,9.1};
    print(v);
    return 0;
}</pre>
```

009

Exemple concret de référence

Fonction de multiplication par un scalaire

```
struct vec4
{
          double x,y,z,w;
};

vec4 mul(vec4 const& v,double s)
{
          vec4 v_out={v.x*s,v.y*s,v.z*s,v.w*s};
          return v_out;
}

int main()
{
          vec4 v1={1.1,7.4,8.8,9.1};
          vec4 v2=mul(v1,3.5);
          std::cout<<v2.x<</pre>
"<<v2.y<<std::endl;
return 0;
}
```

Accesseur

Accesseur de type 'get'

Indique que cette fonction ne modifie pas T

011

010

800

Accesseur Accesseur de type 'set' class vec50 private: float T[50]; public: void init() for(int k=0; k<50; ++k) T[k]=static_cast<float>(k); retourne une référence! float& value(unsigned int i) assert(i<50); return T[i]; }; int main() On peut manipuler la valeur de l'extérieur vec50 v; v.init(); v.value(10)=15; v.value(10)++; Utile pour des vecteurs/matrices! m(4,8)=18;std::cout<<v.value(10)<<std::endl; Equivalent Java: return 0; m.set(4,8,18); ou m.set(18,4,8); 012

Limiter les références non const

Ne pas abuser des modifications de paramètres par référence Difficile à lire

Il faut des règles de programmations

Il faut éviter tout doute sur les variables modiées

- Ex. Documenter le code
 - Références in/out en debut/fin de fonction
 - Variables modifiées passées par pointeurs

014

...

Précautions avec les références

Une référence peut devenir invalide si la variable n'existe plus. (Même problème qu'avec les pointeurs)

```
int& f(int a)
{
   int b=2*a;
   return b;
}

int main()
{
   int a=10;
   int& ref=f(a);
   std::cout<<ref<<std::endl;
   return 0;
}</pre>

   b est une variable locale
   retourne une ref sur une
   variable locale (Warning)

variable locale n'existe plus
   Fuite mémoire, segfault, etc...

return 0;
}
```

013