DAUPHINE
UNIVERSITÉ PARIS

Master Mathématiques, Informatique, Décision, Organisation (MIDO)
Mention MMD, Spécialité ISF 2ème année en apprentissage

Langage C++/VBA – Lien C++/Excel (DLL)

Maude Manouvrier

- Bases du langage C++ (4 séances de 3h)
- VBA sous Excel et DLL (2 séances de 3h)
- Compléments C++ (si le temps …)

http://www.lamsade.dauphine.fr/~manouvri/C++/M2ISF/

Documents en ligne

- *Financial Numerical Recipes in C* de Bernt Arne Ødegaard, 2007
  `http://finance.bi.no/~bernt/gcc_prog/recipes/recipes.pdf`
- *A brief introduction to C++ and Interfacing with Excel* de A. L. Hazel
  `http://www.maths.manchester.ac.uk/~ahazel/EXCEL_C++.pdf`
- VB-VBA : Programmer efficacement Microsoft Excel de *J.M.* Rabilloud
  `http://www.maths.manchester.ac.uk/~ahazel/EXCEL_C++.pdf`
- *Petit manuel de survie pour C++* de François Laroussinie, 2004-2005,
  `http://www.lsv.ens-cachan.fr/~fl/Cours/docCpp.pdf`
- *Thinking in C++* de Bruce Eckel, 2003
  `http://mindview.net/Books/TICPP/ThinkingInCPP2e.html`
- + voir sur **`http://www.cplusplus.com/`**,
  `http://www.librecours.org` et
  `http://www.freetechbooks.com/`

Partie 1 : Langage C++

- Historique
- Premiers pas en C++
- Types
- Déclaration de variables
- Opérations
- Structures de contrôle
- Fonctions
- Pointeurs
- Classes et objets (principales notions)

Historique du Langage C++

- Première version développée par Bjarne Stroustrup de Bell Labs AT&T en 1980
- Appelé à l'origine « Langage C avec classes »
- Devenu une norme ANSI/ISO C++ en juillet 1998 (ISO/IEC 14882)

  ANSI : American National Standard Institute

  ISO : International Standard Organization

- Nouvelle version du standard fin 2009

Premiers pas en C++ (1/7)

```
// Programme affichant la racine carrée d'un entier saisi au clavier
// Directives préprocesseur (#) pour inclure une librairie
#include <iostream> // Librairie standard d'entrée/sortie
#include <cmath> // Librairie d'opérations mathématiques de base
using namespace std; // Espace de nom de la librairie standard

int main() // Programme principal
{
    int i; //Déclaration d'une variable de type entier
    cout << "Saisir un entier : \n"; // Affiche à l'écran
    cin >> i; // Lit un entier saisi au clavier

    if(i>=0) // Si i est positif

     { // Affichage de la racine carrée de i
       cout << "Racine carré = "  << sqrt(i);
     }
    else { //Si i négatif, affichage d'un message d'erreur
           cout << "L'entier n'est pas positif!" << endl;
           // endl = saut de ligne
         }
    return 0;
}
```

Premiers pas en C++ (2/7)

- Utilisation d'espaces de noms (*namespace*) lors de l'utilisation de nombreuses bibliothèques pour éviter les conflits de noms
- Espace de noms : association d'un nom à un ensemble de variable, types ou fonctions

  Ex. Si la fonction *MaFonction()* est définie dans l'espace de noms *MonEspace*, l'appel de la fonction se fait par *MonEspace::MaFonction()*

- Pour être parfaitement correct :

```
std::cin
std::cout
std::endl
```

- Pour éviter l'appel explicite à un espace de noms : **`using`**

```
using std::cout
using namespace std;
```

Premiers pas en C++ (3/7)

- Programme C++ généralement constitué de plusieurs modules, compilés séparément
- Fichier entête – d'extension `.h` (ou `.hh` ou `.hpp`)
  - Contenant les déclarations de types, fonctions, variables et constantes, etc.
  - Inclus via la commande `#include`
- Fichier source – d'extension `.cpp` ou `.cc` ou `.C`

MonFichierEnTete.h

```
#include <iostream>
char* MaChaine;
void MaFonction();
```

MonFichier.cpp

```
#include "MonFichierEnTete.h"
void MaFonction()
{
 cout << MaChaine << " \n " ;
}
```

MonProgPirncipal.cpp

```
#include "MonFichierEnTete.h"
char *MaChaine="Chaîne à afficher";
int main()
{
  MaFonction();
}
```

Premiers pas en C++ (4/7)

- Langage C++ : langage compilé => fichier exécutable produit à partir de **fichiers sources** par **un compilateur**
- Compilation en 3 phases :
  - *Preprocessing* : Suppression des commentaires et traitement des directives de compilation commençant par # => code source brut
  - **Compilation** en fichier objet : compilation du source brut => fichier objet (portant souvent l'extension .obj ou .o sans `main`)
  - **Edition de liens** : Combinaison du fichier objet de l'application avec ceux des bibliothèques qu'elle utilise =>fichier exécutable binaire ou une librairie dynamique (`.dll` sous Windows)
- Compilation => vérification de la syntaxe mais **pas de vérification de la gestion de la mémoire** (erreur d'exécution *segmentation fault*)

Premiers pas en C++ (5/7)
Erreurs générées

- Erreurs de compilation

  Erreur de syntaxe, déclaration manquante, parenthèse manquante,...

- Erreur de liens

  Appel a des fonctions dont les bibliothèques sont manquantes

- Erreur d'exécution

  Segmentation fault, *overflow*, division par zéro

- Erreur logique

Premiers pas en C++ (6/7)
Compilateur

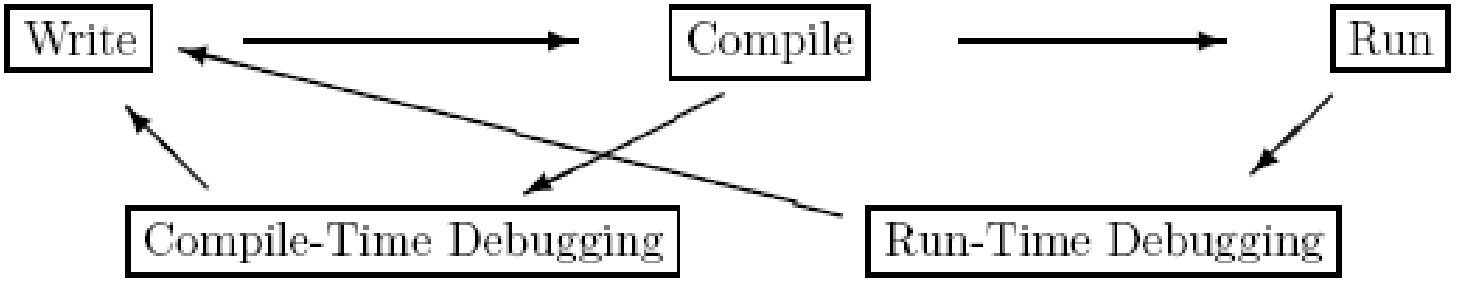
- Compilateurs gratuits (*open-source*) :
  - **Plugin C++ pour Eclipse**
    `http://www.eclipse.org/cdt/`
    Télécharger `http://wascana.sourceforge.net/` pour développer sous Windows
    Dernière version Juin 2009
  - **Dev-C++ 5**
    `http://www.bloodshed.net/devcpp.html`

    Apparemment pas de mise à jour depuis 2005
- Compilateurs propriétaires :
  - Visual C++ (Microsoft – disponible au CRIO INTER-UFR-version gratuite disponible Visual Express mais nécessité de s'inscrire sur le site de Windows `http://msdn.microsoft.com/fr-fr/express/`)
  - Borland C++ (version libre téléchargeable `http://www.codegear.com/downloads/free/cppbuilder`)

Premiers pas en C++ (7/7)
Quelques règles de programmation

1. Définir les classes, inclure les librairies etc. dans un fichier d'extension .h
2. Définir le corps des méthodes et des fonctions, le programme **`main`** etc. dans un fichier d'extension .cpp (incluant le fichier .h)
3. Compiler régulièrement
4. Pour déboguer :
   - Penser à utiliser les commentaires et les **`cout`**
   - Utiliser le débogueur



Types (1/6)
Types de base

```
bool this_is_true = true; // variable boléenne

int i = 0; // entier

long j = 123456789; // entier long

float f = 3.1; // réel

// réel à double précision
double pi = 3.14159265358979323846264643;

char c='a'; // caractère
```

Déclaration et initialisation de variables

Instructions équivalentes :

```
int i(0) ;
long  j(123456789);
…
```

Types (2/6)

Le type d'une donnée détermine :

- La place mémoire (**`sizeof()`**)
- Les opérations légales
- Les bornes

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int (short)`	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int (long)`	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	true or false
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 *or* 4 bytes	1 wide character

Types (3/6) : réel

- Représenté par un nombre à virgule flottante :
  - Position de la virgule repérée par une partie de ses bits (exposant)
  - Reste des bits permettant de coder le nombre sans virgule (mantisse)
- Nombre de bits pour le type **`float`** (32 bits) : 23 bits pour la mantisse, 8 bits pour l'exposant, 1 bit pour le signe
- Nombre de bits pour le type **`double`** (64 bits) : 52 bits pour la mantisse, 11 bits pour l'exposant, 1 bit pour le signe
- Nombre de bits pour le type **`long double`** (80 bits) : 64 bits pour la mantisse, 15 bits pour l'exposant, 1 bit pour le signe
- Précision des nombres réels approchée, dépendant du nombre de positions décimales, d'au moins :
  - 6 chiffres après la virgule pour le type **`float`**
  - 15 chiffres après la virgule pour le type **`double`**
  - 17 chiffres après la virgule pour le type **`long double`**

Types (4/6) : char

- Deux types pour les caractères, codés sur 8 bits/1octets
  - **`char`** (-128 à 127)
  - **`unsigned char`** (0 à 255)

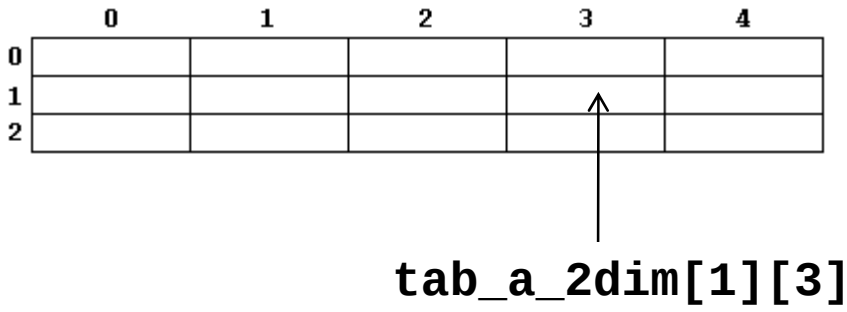
  Exemple: `'a' 'c' '$' '\n' '\t'`
- Les caractères imprimables sont toujours positifs
- Caractères spéciaux :

  `\n` (nouvelle ligne)
  `\t` (tabulation horizontale)
  `\f` (nouvelle page)
  `\b` (*backspace*)
  `EOF`, …

Types (5/6) : tableau

```
int tab1[5] ; // Déclaration d'un tableau de 5 entiers

// Déclaration et initialisation d'un tableau de 3 entiers
int tab2 [] = {1,2,3} ; // Les indices commencent à zéro
```

```
int tab_a_2dim[3][5];
```




```
char chaine[]= "Ceci est une chaîne de caractères";

// Attention, le dernier caractère d'une chaîne est '\0'
char ch[]= "abc" ; // ⇔ char ch[]= {'a','b','c', '\0'};
```

Types (6/6)
Classes / Patrons de classes (*Template*) existants

- **Classe** (Wikipedia) **:** *(on reviendra sur cette notion!!)*
  - En programmation **orientée-objet**, déclaration des propriétés communes à un ensemble d'objets
  - Déclaration des **attributs** représentant l'état des objets et des **méthodes** représentant leur comportement.
- **Patrons de classes (*Template*) :** Définition générique d'une classe permettant au compilateur d'adapter automatiquement la classe à différents types
- Dans la **STL (*Standard Template Library*)** : Définition de structures de données et de fonctions telles que les conteneurs (vecteur, liste), d'algorithmes généraux, etc. - Incluse dans la norme C++ ISO/IEC 14882

```
#include <string> // classe chaîne de caractères
#include <vector> // patron de classes vecteur
#include <list> // patron de classes liste

// chaîne de caractères
string chaine("Ceci est une chaîne de caractères");
// autre initialisation possible : string chaine="Ceci …";

vector<int> v1(4, 99) ; // vecteur de 4 entiers égaux à 99
vector<int> v2(7) ;   // vecteur de 7 entiers

list<char> lc2 ; // Liste de caractères
```

Documentation : `http://www.cplusplus.com/reference/stl/`

Déclaration des variables : Règles d'identification et portée

- Toute variable doit être déclarée avant d'être utilisée
- Constante symbolique : `const int taille = 1000;`

```
// Impossible de modifier taille dans la suite du programme
```

- La portée (visibilité) d'une variable commence à la fin de sa déclaration jusqu'à la fin du bloc de sa déclaration

```
// une fonction nommée f de paramètre i
void f (int i) {
                    int j; // variable locale
                    j=i;
                  }
```

- Toute double déclaration de variable est interdite dans le même bloc

```
int i,j,m; // variable globale
void f(int i) {
                    int j,k; // variable locale
                    char k;  // erreur de compilation
                    j=i;
                    m=0;
                  }
```

Opérations mathématiques de base

```
int i = 100 + 50;
int j = 100 - 50;
int n = 100 * 2;
int m = 100 / 2; // division entière
int k= 100 % 2; // modulo – reste de la division entière

i = i+1;
i = i-1;

j++; // équivalent à j = j+1;
j--; // équivalent à j = j-1;

n += m; // équivalent à n = n+m;
m -= 5; // équivalent à m = m-5;
j /= i; // équivalent à j = j/i;
j *= i+1; // équivalent à j = j*(i+1);

int a, b=3, c, d=3;
a=++b; // équivalent à b++; puis a=b; => a=b=4
c=d++; // équivalent à c=d; puis d++; => c=3 et d=4
```

A utiliser avec parcimonie – car code vite illisible!!

Autres opérations mathématiques

`cmath` : Librairie C++ de fonctions mathématiques standard en trigonométrie (cos, sin …), fonctions exponentielles et logarithmiques, puissance, valeur absolue etc.

Voir documentation :
`http://www.cplusplus.com/reference/clibrary/cmath/`

Exemple :

```
#include <cmath>
double a = pow(7,3); // a=7^3
double b = exp(1); // b=e^1
```

Comparaisons

```
int i,j;
…
if(i==j) // évalué à vrai (true ou !=0) si i égal j
{
  … // instructions exécutées si la condition est vraie
}

if(i!=j) // évalué à vrai (true ou !=0) si i est différent de j

if(i>j) // ou (i<j) ou (i<=j) ou (i>=j)


if(i) // toujours évalué à faux si i==0 et vrai si i!=0
if(false) // toujours évalué à faux
if(true) // toujours évalué à vrai
```

Ne pas confondre = (affectation) et == (test d'égalité)

```
if (i=1) // toujours vrai et i vaut 1 après
```

Opérations du type `char`

- Fonctions d'entrées/sorties de base (**`stdlib.h`**)
  - `c=getchar(); // lit un caractère sur l'entrée standard (clavier)`
  - `putchar(c); // envoie un caractère sur sortie standard (écran)`
- Fonction de qualification de caractères :
  - `isalpha(c) // valeur non nulle si c est une lettre`
  - `isupper(c) // valeur non nulle si c est une lettre majuscule`
  - `isdigit(c) // valeur non nulle si c est un chiffre`
- Fonction de conversion de caractères:
  `toascii, toupper, tolower, atoi, atof, …`
- Voir documentation :
  `http://www.cplusplus.com/reference/clibrary/cctype/`

Opérations sur les chaînes de caractères

- Sur les tableaux de caractères : fonctions de la librairie C **`string.h`**

  Voir documentation : **`http://www.cplusplus.com/reference/clibrary/cstring/`**

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char source[]="chaîne exemple",destination[20];
  strcpy (destination,source); // copie la chaîne source dans la
                                  chaîne destination
}
```

- Sur la classe **`string`** : méthodes appliquées aux objets de la classe **`string`**

  Voir documentation : **`http://www.cplusplus.com/reference/string/string/`**

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
    string str ("chaîne test");
    cout << " str contient " << str.length() << " caractères s.\n";
    return 0;
}
```

On reviendra sur les notions de fonctions et de méthodes!!

Structures de contrôles (1/4)

```
x = 10;
y = x > 9 ? 100 : 200; // équivalent à
                       // if(x>9) y=100;
                       // else y=200;

int main()
{
   float a;

   cout << "Entrer un réel :";
   cin >> a;

   if(a > 0) cout << a << " est positif\n";
   else
    if(a == 0) cout << a << " est nul\n";
    else cout << a << " est négatif\n";
}

// Mettre des {} pour les blocs d'instructions des if/else pour
// éviter les ambiguïtés et lorsqu'il y a plusieurs instructions
```

Structures de contrôles (2/4)

```
for(initialisation; condition; incrémentation)
   instruction; // entre {} si plusieurs instructions
```

Exemples :

```
for(int i=1; i <= 10; i++)
  cout << i << " " << i*i << "\n"; // Affiche les entiers de
                                   // 1 à 10  et leur carré

int main()
{
  int i,j;
  for(i=1, j=20; i < j; i++, j-=5)
   {
     cout << i << " " << j << "\n";
   }
}
```

Résultat affiché :

```
1 20
2 15
3 10
4 5
```

Structures de contrôles (3/4)

```
int main()
{ char ch;
  double x=5.0, y=10.0;
  cout << " 1. Afficher la valeur de x\n";
  cout << " 2. Afficher la valeur de y\n";
  cout << " 3. Afficher la valeur de xy\n";
  cin >> ch;

  switch(ch)
  {
    case '1': cout << x << "\n";
              break; // pour sortir du switch
                     // et ne pas exécuter les commandes suivantes
    case '2': cout << y << "\n";
              break;
    case '3': cout << x*y << "\n";
              break;
    default: cout << « Option non reconnue\n";

  } \\ Fin du switch

} \\ Fin du main
```

Structures de contrôles (4/4)

```
int main ()
{
   int n=8;
   while (n>0)
   { cout << n << " ";
      --n;
   }
    return 0;
 }
```

Instructions exécutées tant que *n* est supérieur à zéro

Résultat affiché :

```
8 7 6 5 4 3 2 1
```

```
int main ()
{
   int n=0;
   do
   { cout << n << " ";
      --n;
   }
   while (n>0);
   return 0;
 }
```

Instructions exécutées une fois puis une tant que *n* est supérieur à zéro

Résultat affiché :

```
0
```

Fonctions (1/5)

```
type_retour nom_de_la_fonction( arguments ) // signature
 { corps de la fonction }
```

- Appel de fonction toujours précédé de la déclaration de la fonction sous la forme de prototype (signature)
- Une et une seule définition d'une fonction donnée mais autant de déclaration que nécessaire

Exemples :

```
int carre(int a)
{
  return(a*a);
}
```

```
float racine(float nbr)
{
  float racine=.0; // variable locale
  while(racine*racine<nbr)
    racine+=.01;
  return(racine); // valeur retournée
}
```

```
void afficheProduit(int i, int j) // fonction ne retournant rien
{ cout << i*j;
}
```

Fonctions (2/5)

Attention au passage des paramètres

Passage des paramètres par valeur : par défaut en C++

```
void echange(int,int); // Déclaration (prototype) de la fonction
                       // A mettre avant tout appel de la fonction

int main()
{  int n=10, p=20;
   cout << "avant appel: n="
        << n << " p=" << p << endl;
   echange(n,p);
   cout << "apres appel: n=" << n
        << " p=" << p << endl;
}
```

Lors de l'appel **echange(n,p)**:
a prend la valeur de **n**
et **b** prend la valeur de **p**
Mais après l'appel (à la sortie de la fonction), **n** et **p** restent inchangées

```
void echange(int a, int b) // Définition de la fonction
{  int c;
   cout << "debut echange : a=" << a
        << " b=" << b << endl;
   c=a; a=b; b=c;
   cout << "fin echange : a=" << a
        << " b=" << b << endl;
}
```

```
avant appel: n=10 p=20
debut echange: a=10 b=20
fin echange: a=20 b=10
apres appel: n=10 p=20
```

Fonctions (3/5)

Passage des paramètres par référence :

On reviendra sur la notion d'emplacement mémoire!!

```
void echange(int&,int&); // '&' désigne l'emplacement mémoire

int main() // le main est inchangé
{ int n=10, p=20;
  cout << "avant appel: n="
       << n << " p=" << p << endl;
  echange(n,p); // l'appel est inchangé
  cout << "apres appel: n=" << n
       << " p=" << p << endl;
}
```

Lors de l'appel **`echange(n,p)`**:
`a` et **`n`** correspondent au même emplacement mémoire
idem pour **`b`** et **`p`**
Donc après l'appel (à la sortie de la fonction), les valeurs de **`n`** et **`p`** sont modifiées

```
void echange(int& a, int& b) // modif. de la signature uniquement
{ int c;
  cout << "debut echange : a=" << a
       << " b=" << b << endl;
  c=a; a=b; b=c;
  cout << "fin echange : a=" << a
       << " b=" << b << endl;
}
```

```
avant appel: n=10 p=20
debut echange: a=10 b=20
fin echange: a=20 b=(10
apres appel: n=20 p=10
```

Pas de conversion possible : **`echange(n,p);`** avec **`n et p`** de type **`float`**
et pas d'appel avec une constante : **`echange(3,4);`**

Fonctions (4/5)

- Possibilité de surcharger ou sur-définir une fonction

```
int racine carree (int x) {return x * x;}

double racine_carree  (double y) {return y * y;}
```

- Possibilité d'attribuer des valeurs par défaut aux arguments

```
void MaFonction(int i=3, int j=5); // Déclaration
int x =10, y=20;

MaFonction(x,y); // Appel avec 2 argument

MaFonction(x);   // Appel avec 1 argument <=> MaFonction(x,5);

MaFonction();    // Appel sans argument <=> MaFonction(3,5);
```

Les arguments concernés doivent obligatoirement être les derniers de la liste

A fixer dans la déclaration de la fonction pas dans sa définition

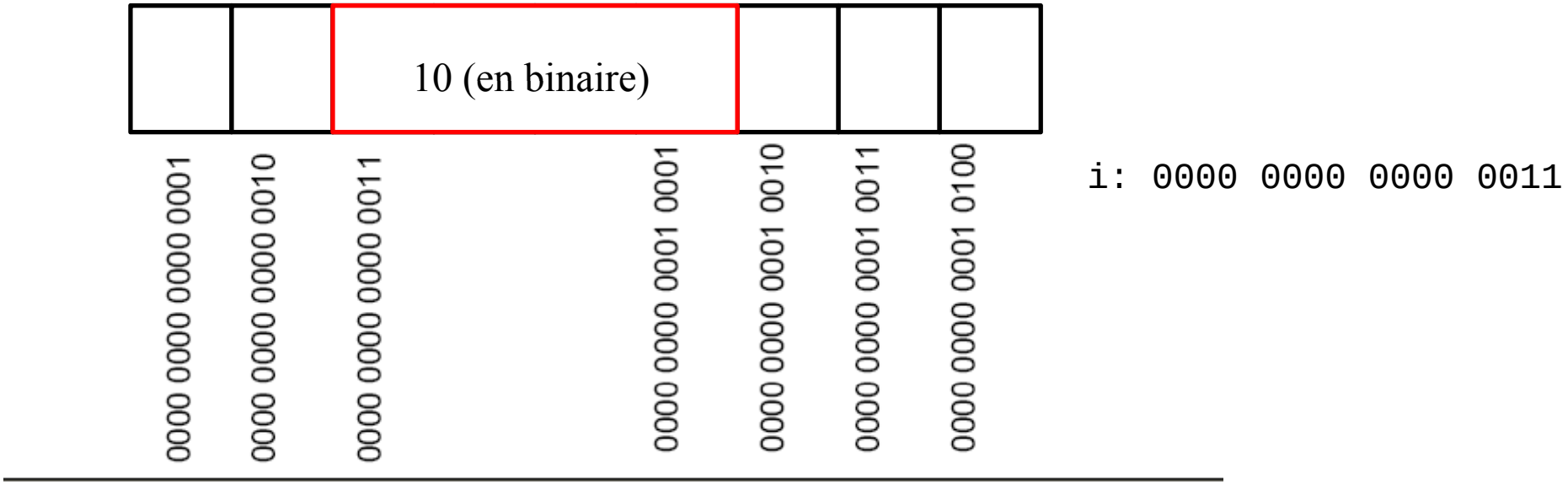
Fonctions (5/5)

Fonction récursive : fonction qui s'appelle elle-même

```
#include <iostream>
using namespace std;
long factorial (long a)
{
  if (a > 1) // condition d'arrêt de l'appel récursif
     return (a * factorial (a-1)); // appel récursif
  else return (1);
}
int main ()
{
  long n;
  cout << "Entrer un nombre: ";
  cin >> n;
  cout << n << "! = " << factorial (n);
  return 0;
}
```

Pointeurs (1/8)

Mémoire décomposée en "cases" (1 octet) consécutives numérotées (ayant une adresse) que l'on peut manipuler individuellement ou par groupe de cases contigües



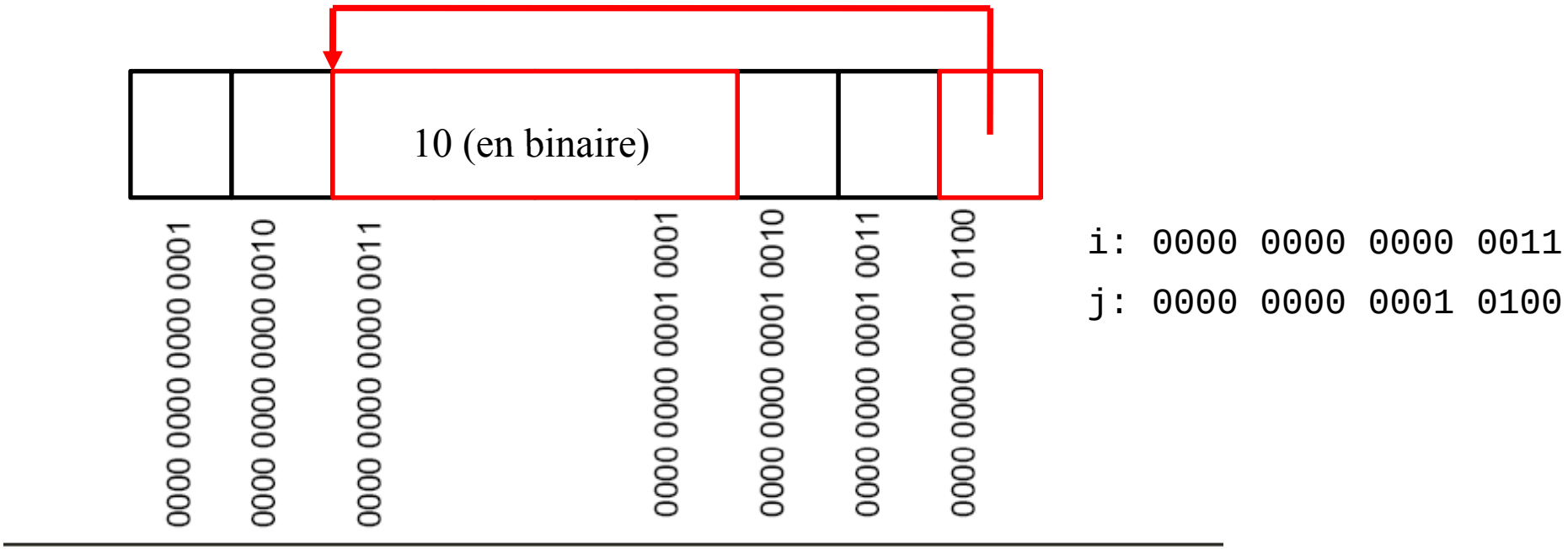

```
int i=10 ;
```

1. Réservation d'une zone mémoire de 4 octets (la 1ère libre)
2. Association du nom i à l'adresse du début de la zone
3. Copie de la valeur en binaire dans la zone mémoire

&i correspond à l'adresse du début de la zone mémoire où est stockée la valeur de **i**

Pointeurs (2/8)

Pointeur = variable contenant une adresse en mémoire




```
int i=10;
int *j=&i;
```

Pointeurs (3/8)

- Allocation dynamique de la mémoire :

```
// Déclaration d'un pointeur p sur réel
// et allocation d'une zone en mémoire de la taille d'un réel
// dont l'adresse est stockée dans p
float *p = new float;

// Déclaration d'un pointeur stockant l'adresse d'une zone
// mémoire pouvant contenir 20 entiers (20*sizeof(int) octets)
int *tab = new int[20];
```

- **`new type`** : définition et allocation d'un pointeur de type ***`type*`***
- **`new type [n]`** : définition d'un pointeur de type ***`type*`*** sur un tableau de *n* éléments de type ***`type`***
- En cas d'échec de l'allocation, **`new`** déclenche une exception du type `bad_alloc`

```
int main ()
 { try { int *p = new int ; }
   catch (bad_alloc) // Si le new s'est mal passé
   { cout << "exception : le new a échoué" << endl ;
     exit (-1) ;
   }
 }
```

Pointeurs (4/8)

- Manipulation de la valeur pointée :

```
int *p = new int; // p est de type int*
(*p)=3; // (*p) est de type int
int *tab = new int [20]; // tab est de type int*
// tab correspond à l'adresse du début de la zone mémoire
// allouée pour contenir 20 entiers
(*tab)=3; // équivalent à tab[0]=3
```

- Manipulation de pointeur :

```
tab++; // décalage de tab d'une zone mémoire de taille sizeof(int)
       // tab pointe sur la zone mémoire devant contenir le 2ème
       // élément du tableau
(*tab)=4; // équivalent à tab[1]=4
          // car on a décalé tab à l'instruction précédente
*(tab+1)=5; // équivalent à tab[2]=5 sans décaler tab
tab-=2 ; // tab est repositionné au début de la zone mémoire
         // allouée par new
```

- Libération de la mémoire allouée :

```
delete p;
delete [] tab;
```

Ne pas oublier de libérer la mémoire allouée

Pointeurs (5/8)

Manipulation des pointeurs et des valeurs pointées (suite)

```
int *p1 = new int;
int *p2 = p1 ; // p2 pointe sur la même zone mémoire que p1
*p2=11; //=> *p1=11; car p1 et p2 pointe sur la même zone mémoire
int *tab = new int [20];
*tab++=3; // équivalent à *(tab++)=tab[1]=3
          // car ++ a une priorité plus forte que *
          // ⇔ tab++; *tab=3;
(*tab)++; // => tab[1]=4
int i=12;
p2=&i; // p2 pointe sur la zone mémoire où est stockée i
*p2=13; // => i=13
p2=tab; // p2 pointe comme tab sur le 2ème élément du tableau
p2++; // p2 pointe sur le 3ème élément (d'indice 2)
*p2=5; // => tab[2]=5 mais tab pointe toujours sur le 2ème élément
p1=p2; // => p1 pointe sur la même zone que p2
       // ATTENTION : l'adresse de la zone allouée par new pour p1
       //             est perdue!!
```

Pointeurs (6/8)

Passage des paramètres de fonction par valeur :

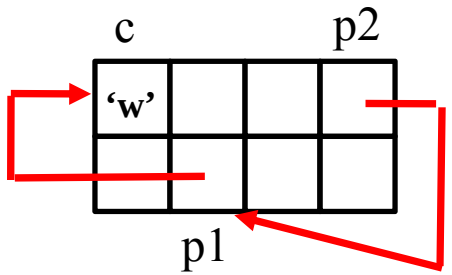
```
void echange(int*,int*);
int main()
{
   int n=10, p=20;
   cout << "avant appel: n=" << n
        << " p=" << p << endl;
   echange(&n,&p); // les adresses de n et p sont passées en
                   // paramètre pour pouvoir modifier leurs valeurs
   cout << "apres appel: n=" << n
        << " p=" << p << endl;
}
void echange(int* a, int* b) // arguments de type pointeur
{
   int c;
   cout << "debut echange : *a=" << *a
       << " *b=" << *b << endl;
   c=*a; *a=*b; *b=c; // échange des valeurs pointées
   cout << "fin echange : *a=" << *a
        << " *b=" << *b << endl;
}
```

```
avant appel: n=10 p=20
debut echange: *a=10 *b=20
fin echange: *a=20 *b=(10
apres appel: n=20 p=10
```

Pointeurs (7/8)

- Pointeurs de pointeur :

```
char c='w';
char *p1=&c; // p1 a pour valeur l'adresse de c
             // (*p1) est de type char
char **p2=&p1; // p2 a pour valeur l'adresse de p1
               // *p2 est de type char*
               // **p2 est de type char
```




```
cout << c ;
cout << *p1;
cout << **p2;
```
// 3 instructions équivalentes qui affiche 'w'

- Précautions à prendre lors de la manipulation des pointeurs :
  - Allouer de la mémoire (**new**) ou affecter l'adresse d'une zone mémoire utilisée (**&**) avant de manipuler la valeur pointée
  - Libérer (**delete**) la mémoire allouée par **new**
  - Ne pas perdre l'adresse d'une zone allouée par **new**

Pointeurs (8/8)

Arguments de la fonction main : pour passer des paramètres au programme

```
// argc = argument count et argv = argument value
int main( int argc, char* argv[] )
{
  cout << "Le nom du programme est : " << argv[0] << endl;
  cout << « Il y a " << argc-1 << " arguments" << endl;
  for ( int i = 1; i < argc; ++i )
   cout << argv[i] << endl;
  return 0;
}
```

Classes et objets (1/11) : définitions

- Classe :
  - Regroupement de données (attributs ou champs) et de méthodes (fonctions membres)
  - Extension des structures (`struct`) avec différents niveaux de visibilité (`protected`, `private` et `public`)
- En programmation orientée-objet pure: encapsulation des données et accès unique des données à travers les méthodes
- Objet : instance de classe
  - Attributs et méthodes communs à tous les objets d'une classe
  - Valeurs des attributs propres à chaque objet
- Encapsulation
  - Caractérisation d'un objet par les spécifications de ses méthodes : interface
  - Indépendance vis à vis de l'implémentation

Classes et objets (2/11) : 1er exemple de classe

```
#include <string>
using namespace std;

class Account
{ public: // Attributs publics (visibles de partout)
        string Name;
        double Balance;
};

int main() "
{
  // Déclaration de deux objets de la classe Account
  Account ac_001, ac_002;

  // Initialisation des attributs des objets
  ac_001.Name = "John Smith"; ac_001.Balance = 0.0;
  ac_002.Name = "Jane Jones"; ac_002.Balance = 10.0;
}
```

Classes et objets (3/11) : 1er exemple de classe

```
class Account
{
  public:
     string Name;
     double Balance;

    // Méthode de la classe Account
    void print_balance()
    {
       cout << Name << "'s account has a balance of "
            << Balance << endl;
    }
};
int main()
{
  Account ac_001;
  ac_001.Name = "John Smith"; ac_001.Balance = 100.0;
  ac_001.print_balance(); // Appel de la méthode pour l'objet
                          // ac_001
}
```

Une méthode s'applique à un objet

Classes et objets (4/11) : 1er exemple de classe

```
class Account
{
  public:
     string Name;
     double Balance;

    // Méthode de la classe Account
    void print_balance()
    {
       cout << Name << "'s account has a balance of "
            << Balance << endl;
    }
    // Prototype de la méthode - le corps peut être défini en
    // dehors de la définition de la classe
    void deposit(const double &);
};
// Définition de la méthode deposit de la classe Account (Account::)
void Account::deposit(double amount)
{
  Balance += amount;
}
```

Une méthode s'applique à un objet

Classes et objets (5/11) : 1er exemple de classe

La bonne manière de définir une classe !!

```
class Account
{
  private: // définir les attributs en privé => non manipulable en
           // dehors du corps des méthodes de la classe
     string Name;
     double Balance;

  public:
   void SetName(string s){ Name = s; }
   void SetBalance(double b){ Balance = b ; }
   string GetName() {return Name;}
   string GetBalance(); {return Balance;}

   …
};
int main()
{
  Account ac_001;
  ac_001.SetName("John Smith“); ac_001.SetBalance(100.0);
}
```

Pour affecter des valeurs aux attributs privés

Pour accéder aux valeurs des attributs privés

Classes et objets (6/11)

2 méthodes particulières

- **Constructeur** : Méthode appelée implicitement à la création d'un objet pour initialiser les valeurs des attributs
  - De même nom que le nom de la classe
  - Par défaut un constructeur sans paramètre qui crée un objet sans initialiser ses valeurs
  - Méthode non typée, pouvant être redéfinie et surchargée => possibilité d'avoir plusieurs constructeurs
- **Destructeur** : Méthode appelée implicitement à la disparition d'un objet
  - De même nom que la classe mais précédé d'un tilde (~)
  - Méthode non typée et sans paramètre pouvant être redéfinie
  - Ne pouvant pas être surchargé => un seul destructeur par classe

Classes et objets (7/11)

```
class Account
{
  private:
     string Name;
     double Balance;
  public:
   // Redéfinition du constructeur par défaut
   Account() {Balance=0.0;}
   // Surcharge du constructeur
   Account(string n) {Name=n; Balance=0.0}
   // Rédéfinition du desctructeur par défaut
   ~Account()
    { cout << "Closing account."; }

  …
};      int main()
        {
          Account ac_001; // Appel du constructeur par défaut
          Account ac_002("John Smith"); // Appel de Account(string n)
        } // A la sortie du main :
          // Appel du destructeur pour ac_001 et ac_002
```

Classes et objets (8/11)

- **Héritage** : Définition de nouvelles classes « dérivées » à partir d'une classe existante « de base »
- **Super-classe** ou classe mère
- **Sous-classe** ou classe fille : spécialisation de la super-classe - héritage des propriétés de la super-classe
- Exemple :

```
// La classe SavingsAccount dérive de Account
// SavingsAccount est une sous-classe de Account
class SavingsAccount : public Account
{
   private: // Attributs spécifiques de la sous-classe
    double Interest_rate;

   public: // Méthodes spécifiques de la sous-classe
     void add_interest()
    { SetBalance(GetBalance()*Interest_rate); }
};
```

Un objet de la classe **`SavingsAccount`** a trois attributs **`Name`**, **`Balance`** et **`Interest_rate`**

Classes et objets (9/11)

```
class SavingsAccount : public Account
{
   private:
     double Interest_rate;

   public:
   // Le constructeur par défaut de la classe fille doit appeler le
   // constructeur de la classe mère
   SavingsAccount() : Account()
   {
     Interest_rate = 0.05;
   }
   // : Account(initial_balance) => appel du constructeur de Account
   SavingsAccount(double initial_balance) : Account(initial_balance)
   {
     //Set the interest rate
     Interest_rate = 0.05;
   }

};
```

L'appel du destructeur de la classe mère est automatique !!

Classes et objets (10/11)

```
class Account
{
  protected:
     string Name;     // pour autoriser les sous-classes a manipuler les
     double Balance; // attributs qui restent privés pour le reste du
                     // programme (i.e. en dehors du corps des méthodes
                     // des sous-classes de Account)
  public:
    …
};

class SavingsAccount : public Account
{
   private:
     double Interest_rate;

   public:

   …
   void add_interest()
        { Balance+=Balance*Interest_rate; }
         // si private dans Account alors il faut utiliser les
        // méthodes publiques SetBalance et GetBalance
};
```

Classes et objets (11/11)

```
Account *p; // Déclaration d'un pointeur sur un objet

p = new Account (1000); // *p est un objet Account dont Balance=1000
                        // => Appel du constructeur de Account

p->SetName("John Smith"); // équivalent à (*p).SetName("John Smith");

p->print_balance(); // équivalent à (*p).print_balance();

delete p; // => appel du destructeur de Account
```

Pointeur particulier à utiliser dans les méthodes : `this`

```
Account::print balance()
{
   cout << this->Name << "'s account has a balance of "
        << this->Balance << endl;
}
```

Utilisation de **`this`** en particulier quand la méthode retourne l'objet courant

Partie 2 : DLL, VBA et Excel

- **Objectif et définitions**
- **Développement d'une DLL C++ pour VBA**
- **Exemple de DLL C++**
- **Exemple de programme C++ utilisant une DLL de manière dynamique**
- **Exemple de programme C++ utilisant une DLL de manière statique**
- **Créer un classeur Excel appelant des fonctions d'une DLL C++**
- **Exemple de fichier Excel utilisant une DLL C++**
- **Exemple de DLL C++ contenant une classe**
- **Comment utiliser une classe de DLL C++ en VBA**
- **Compléments VBA pour Excel**

Objectif

- Objectif (pour ce cours) : Utiliser des fonctions C++ au sein d'un tableau Excel
- Comment ?
  - En créant une bibliothèque DLL (*Dynamic Link Library*) de fonctions C++
  - En appelant les fonctions de cette bibliothèque dans un tableau Excel via des commandes écrites en VBA (*Visual Basic for Application*)

Définitions

- ***Librairy* ou bibliothèque** : fichier contenant plusieurs fonctions d'un programme
- **DLL** (*Dynamic Link Library*) ou bibliothèque liée dynamiquement (Windows) :
  - **Compilée**, donc prête à être utilisée, **chargée dynamiquement en mémoire** lors du démarrage d'un programme (i.e. n'étant pas incluse dans le programme exécutable)
  - Indépendante du(des) programmes qui l'utilise(nt) et pouvant être utilisée par plusieurs programmes en même temps
  - Stockée une seule fois sur le disque dur
  - Permettant aux développeurs (i) de distribuer des fonctions réutilisables par tout programme, sans en dévoiler les sources, (ii) de réduire la taille des exécutables et (iii) de mettre à jour les librairie indépendamment des programmes les utilisant
- **VBA** :
  - Implémentation de *Microsoft Visual Basic* intégrée notamment dans toutes les applications de Microsoft Office
  - Permettant d'écrire des macros sous les applications Office notamment

  - Plus développé ni maintenu par Microsoft depuis juillet 2007 en vue de son remplacement par *Visual Studio Tools for Applications* (VSTA)
  - Toujours utilisé néanmoins dans Office 2007, même si VSTA est disponible

Développement d'une DLL C++

- Structure d'une DLL : Code exécutable en vue d'être appelé par un programme externe
- Table d'exportation de symboles (*Symbols Export Table*) : liste de toutes les fonctions exportées (donc disponibles par un programme externe) ainsi que de leur point d'entrée (adresse du début du code de la fonction)
- Pour développer une DLL sous Visual C++ :
  - Créer un nouveau projet de type *Win32 Dynamic Link Library*
  - Choisir un projet de type « *Empty project* »
  - Créer 3 fichiers :
    - `nom_fichier.h` : contenant les inclusions de bibliothèques (au minimum la bibliothèque standard et `windows.h)` et la définition des fonctions de la bibliothèque
    - `Nom_fichier.cpp` : fichiers contenant le corps des fonctions (et incluant le fichier `nom_fichier.h)`
    - `Nom_fichier.def`: fichier de définition de module pour définir les fonctions du `.cpp` exportées (i.e. utilisables par d'autres programmes – ex. depuis Excel)
- DLL existantes : `http://msdn.microsoft.com/en-us/library/ms723876(vs.85).aspx`

Exemple de DLL C++ (1/4)

- Fichier **MaDll.h**:

```
#include <iostream>
#include <cmath>
#include <windows.h>

// Fonctions exportées de la DLL
extern "C" _declspec(dllexport) double _stdcall carre(double& arg);
extern "C" _declspec(dllexport) double _stdcall
   addition(double& arg1, double& arg2);
extern "C" _declspec(dllexport) double _stdcall
    soustraction(double arg1, double arg2);

// Fonction d'entrée de la DLL
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD ul_reason_for_call,
                       LPVOID lpReserved)
{
   return TRUE;
}
```

- **_stdcall** pour les fonctions C++ devant être utilisées sous Excel
- **extern "C" _declspec(dllexport)** pour pouvoir tester les fonctions dans des programmes C++

Exemple de DLL C++ (2/4)

- Fichier **MaDll.cpp**:

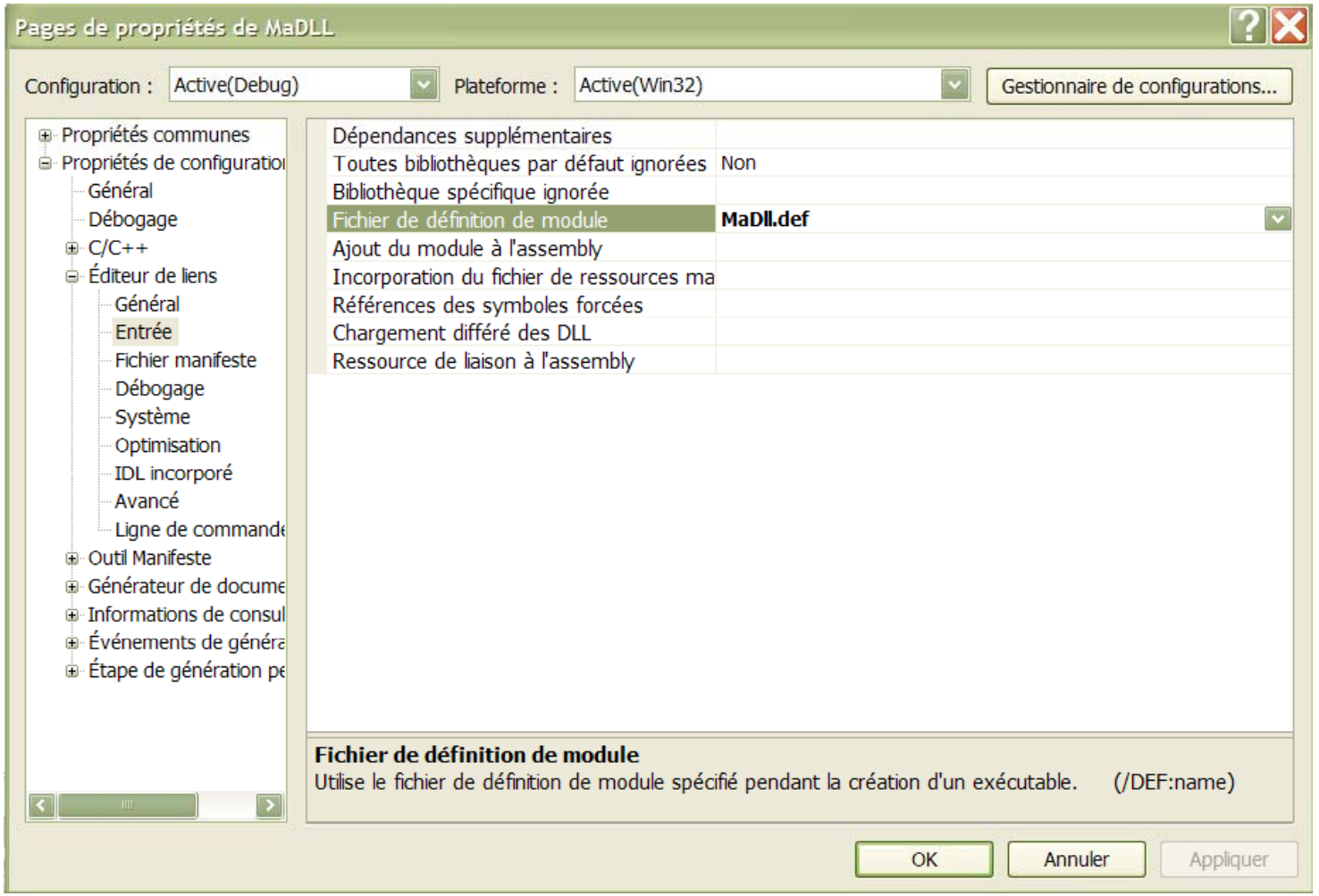
```
#include "MaDll.h"

double _stdcall carre (double& arg)
  {return arg*arg;}

double _stdcall addition (double& arg1, double& arg2)
  {return arg1+arg2;}

double _stdcall soustraction (double arg1, double arg2)
  {return arg1-arg2;}
```

- Fichier **MaDll.def**:

```
LIBRARY MaDll
EXPORTS
        carre
        addition
        soustraction
```

Exemple de DLL C++ (3/4)

Bien spécifier **`MaDll.def`** dans la propriété *Fichier de Définition de Modules* de l'*Editeur de Liens* (obtenu par un clic droit de la souris sur le nom du projet)



Exemple de DLL C++ (4/4)

- Fichiers générés :
  - `Nom_Dll.dll` : fichier de la bibliothèque dynamique
  - `Nom_Dll.lib :` fichier permettant de faire la liaison avec la bibliothèque (i.e. pour qu'un programme puisse accéder aux fonctions de la bibliothèques)
- Possibilité de lier la bibliothèque au programme C++ l'utilisant :
  - De manière statique : déclaration explicite dans le programme (via `#include`) et résolution de liens effectuée par l'éditeur de lien au moment de la phase de compilation du programme
  - De manière dynamique : demande explicite du chargement d'une bibliothèque durant l'exécution du programme
- Sous linux : bibliothèque dynamique d'extension `.so`

Exemple programme C++ utilisant une DLL de manière dynamique

```
// Définition d'un type pointeur sur fonction
typedef double (_stdcall * importFunction)(double&,double&);

int main(void)

{   // Déclaration d'une variable de type pointeur sur fonction
    importFunction AddNumbers;

    double r=5,d=7,result;

    // Chargement de la DLL
    HINSTANCE hinstLib = LoadLibrary(TEXT("C:\\Chemin_d_acces\\MaDLL.dll"));

    // Si le chargement s'est mal passé!
    if (hinstLib == NULL) { cout << "ERROR: unable to load DLL\n";
       return 1;
       }

    // Récupération de la fonction de la librairie via le pointeur
    AddNumbers = (importFunction)GetProcAddress(hinstLib, "addition");

    // Si la récupération de la fonction s'est mal passée!
    if (AddNumbers == NULL) { cout << "ERROR: unable to find DLL function\n";
      FreeLibrary(hinstLib); // Libération de l'espace de chargement de la DLL
      return 1;
     }
    // Appel de la fonction
    result = AddNumbers(r,d);

    FreeLibrary(hinstLib); // Libération de l'espace de chargement de la DLL

    cout << result << endl;

}
}
```

Ne pas oublier d'inclure `<iostream>` *et* `<windows.h>` *!!*

Exemple programme C++ utilisant une DLL de manière statique (1/4)

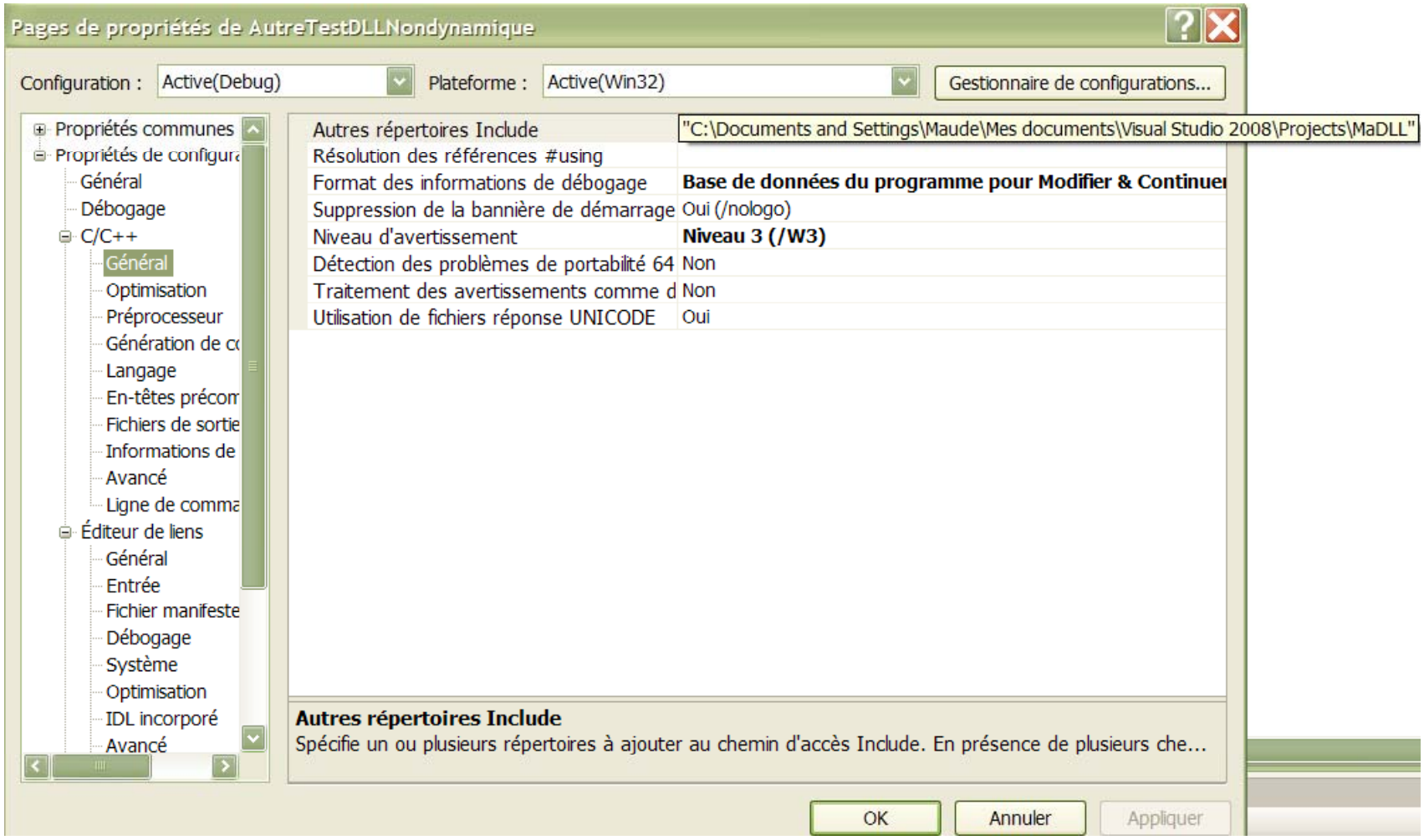
```
#include "MaDll.h"   // Inclusion du .h de la DLL
using namespace std;
int main()
{
    double r,d;
    cin >> r;
    cin >> d;
    cout <<"carre(" << r << ")=" << carre(r) << endl;
    cout <<"addition(" << r << ","<< d << ")=" << addition(r,d) << endl;
}
```

Pour que ça compile et que cela tourne, indiquer dans les propriétés du projet :

- Où trouver le `.h` de la bibliothèque (dans *C/C++/Général/Autres Répertoires Include* – avec des guillemets)
- Où trouver le `.lib` de la bibliothèque (dans *Editeur de Liens/Entrée/Dépendances Supplémentaires*)
- Où trouver le `.dll` de la bibliothèque (dans *Débogage/Environnement* taper *`PATH=chemin_acces_au_fichier_dll`*)

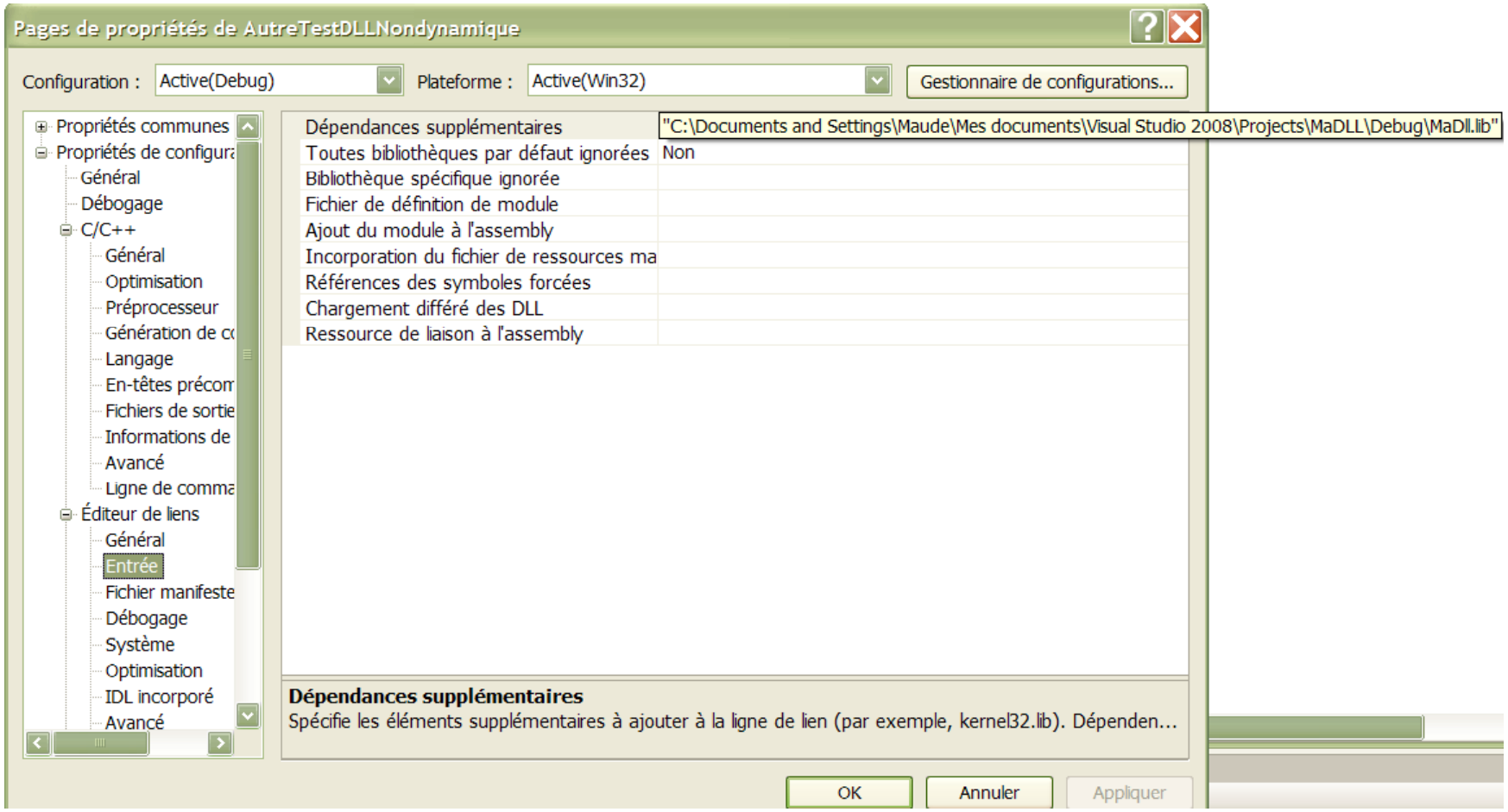
Exemple programme C++ utilisant une DLL de manière statique (2/4)

- Où trouver le .h de la bibliothèque :



Exemple programme C++ utilisant une DLL de manière statique (3/4)

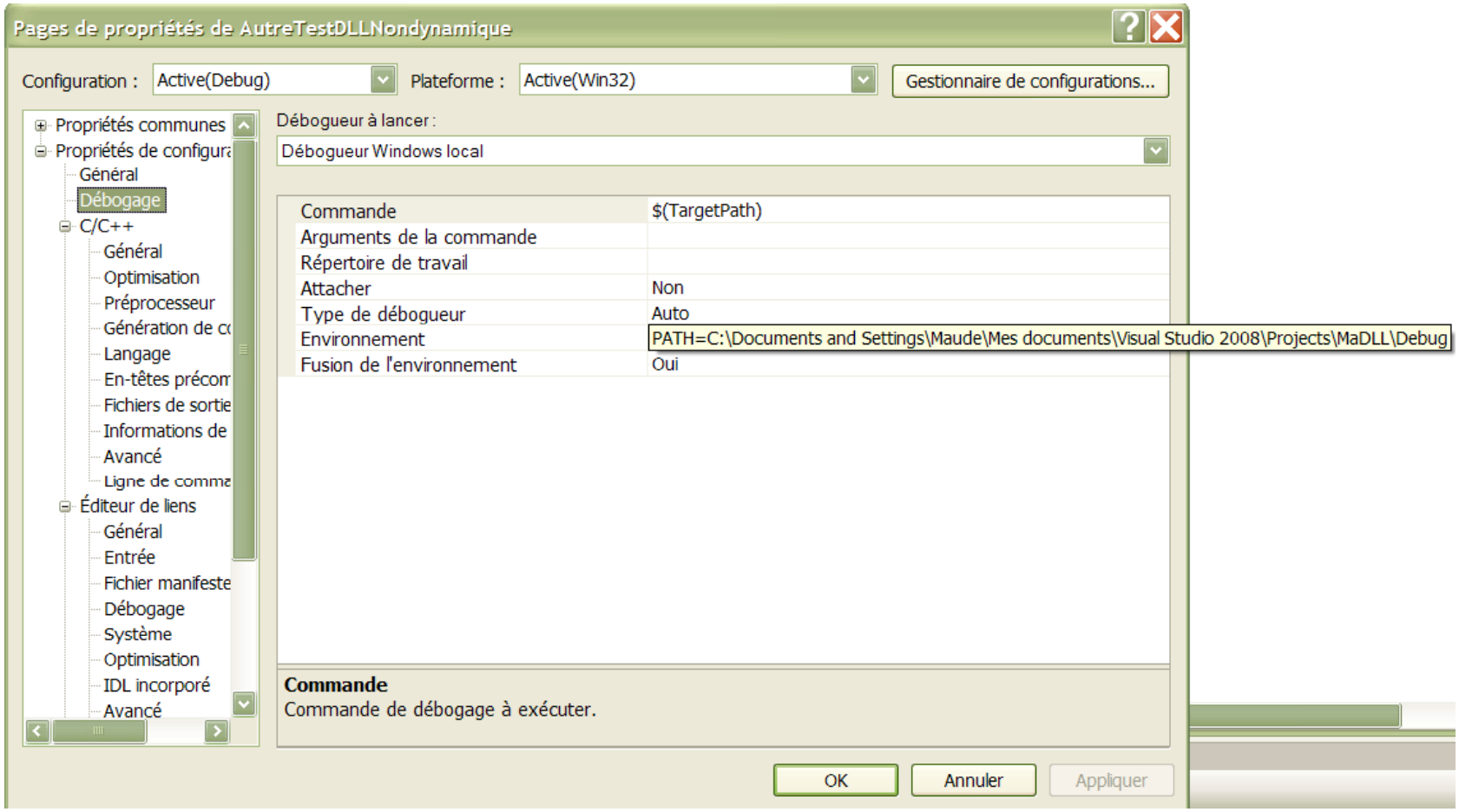
- Où trouver le `.lib` de la bibliothèque :



Bien spécifier le chemin d'accès au fichier `.lib` en mettant des guillemets !!

Exemple programme C++ utilisant une DLL de manière statique (4/4)

- Où trouver le `.dll` de la bibliothèque :



Créer un fichier Excel utilisant une DLL

- Créer un classeur Excel
- Faire *Alt* + *F11* pour ouvrir l'éditeur Visual Basic
- Faire *Alt* + *I* + *M* pour créer un Module
- Ecrire dans le module pour chaque fonction de la DLL devant être utilisée dans le fichier Excel, une commande VBA de la forme :

```
Private Declare Function NomFonction _
Lib "C:\Chemin_d_acces\NomDll.dll" _
(arg1 As Type, arg2 as Type, …) As Type
```

ou sur une seule ligne (sans les _)

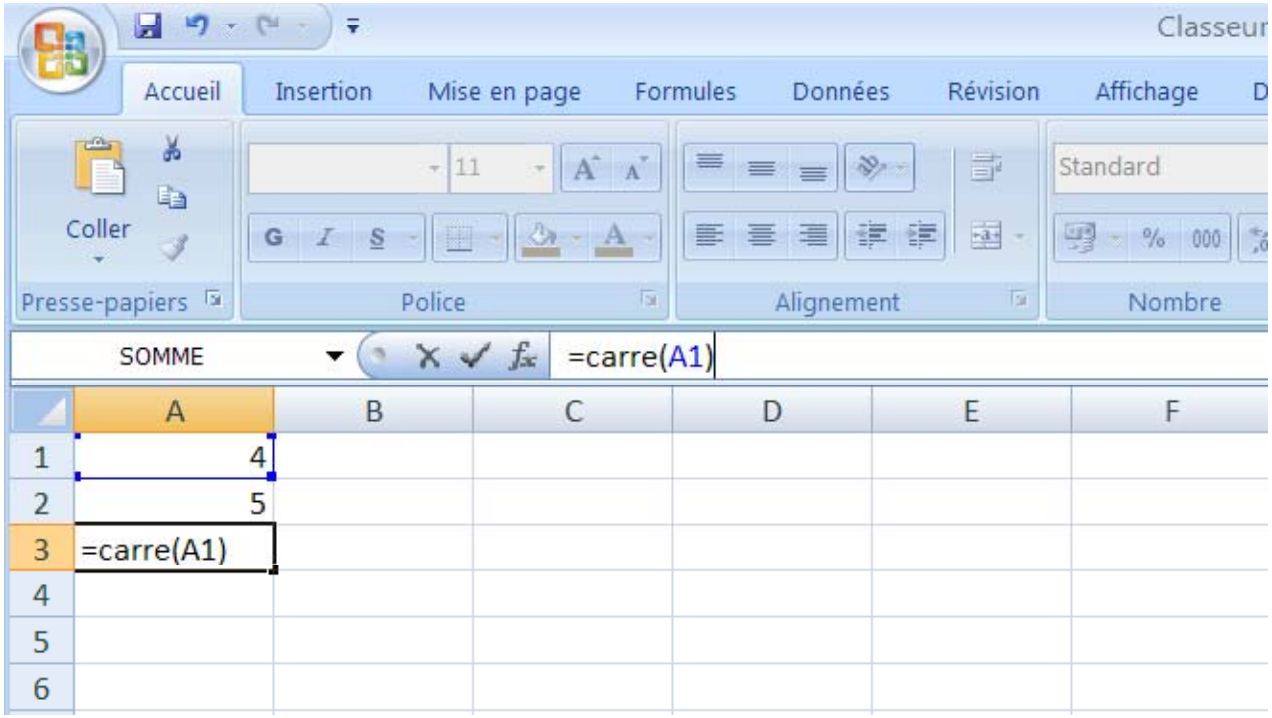
```
Private Declare Function NomFonction Lib
  "C:\Chemin_d_acces\NomDll.dll"(arg1 As Type, arg2
  as Type, …) As Type
```

Sauvegarder le fichier en Fichier Excel ***(prenant en charge les macros)***

Exemple de fichier Excel utilisant une DLL

```
Private Declare Function carre _
Lib "C:\Chemin_d_acces\MaDll.dll"(arg As Double) As Double

Private Declare Function addition _
Lib "C:\Chemin_d_acces\MaDll.dll" _
(arg1 As Double, arg2 As Double) As Double

Private Declare Function soustraction
Lib "C:\Chemin_d_acces\MaDll.dll" _
(ByVal arg1 As Double, ByVal arg2 As Double) As Double
```

Dans le module VBA (fenêtre *Microsoft Visual Basic*)



Dans le classeur Excel

Exemple de DLL C++ contenant une classe (1/2)

- Fichier **MaDll.h**:

```
#include <iostream>
#include <windows.h>
using namespace std;

class ClasseDynamique
{
  public:
    _declspec(dllexport) ClasseDynamique();
     declspec(dllexport) ~ClasseDynamique();
     void _declspec(dllexport) SetAttribut(int a);
    int _declspec(dllexport) GetAttribut();
   void _declspec(dllexport) Afficher();
  private:
    int attribut;
};
```

Classe C++ pouvant être utilisée dans un programme C++ liée à la DLL de manière statique ou dynamique

extern "C" _declspec(dllexport) pour les méthodes de la classe devant être exportées

Exemple de DLL C++ contenant une classe (2/2)

- Fichier **`MaDll.cpp`**:

```
#include "MaDll.h"
ClasseDynamique::ClasseDynamique() { attribut=0; }

ClasseDynamique::~ClasseDynamique() {}

int ClasseDynamique::GetAttribut() { return attribut; }

void ClasseDynamique::SetAttribut(int a) { attribut=a; }

void ClasseDynamique::Afficher()
  { cout << "attribut=" << attribut << endl; }
```

Ne rien mettre dans le `.def` – on verra plus tard comment utiliser cette classe en VBA

- Dans le programme utilisant la DLL (de manière statique par exemple) :

La DLL peut être liée de manière statique ou dynamique

```
#include "MaDll.h"
int main()
{  ClasseDynamique o;
    o.SetAttribut(5);
    o.Afficher();
}
```

Comment utiliser une classe de DLL C++ en VBA (1/5)

- VBA pas objet (possible en VBA.NET – hors programme cette année et pas de lien avec Office)
- Possibilité de « manager » les objets via des fonctions (en utilisant des pointeurs)
- Pour utiliser une classe de DLL C++ dans un programme VBA :
  - Définir dans la DLL C++ des fonctions manipulant les objets de la classe via des pointeurs en les définissant avec **`_stdcall`**
  - Exporter ces fonctions dans le **`.def`**
  - Utiliser ces fonctions dans des modules VBA

Comment utiliser une classe de DLL C++ en VBA (2/5)

- Exemple de **`fichier.h`** d'une classe de DLL C++ - en vue d'une utilisation en VBA :

```
#include <iostream>
#include <windows.h>
using namespace std;

class ClasseDynamique
{
  public:
    _declspec(dllexport) ClasseDynamique();
    _declspec(dllexport) ~ClasseDynamique();
     void _declspec(dllexport) SetAttribut(int a);
    int  declspec(dllexport) GetAttribut();
    void _declspec(dllexport) Afficher();
  private:
    int attribut;
};

extern "C" _declspec(dllexport)
  ClasseDynamique * _stdcall ClasseDynamique_Constr();

extern "C" _declspec(dllexport)
  int _stdcall ClasseDynamique_Destr(ClasseDynamique* Objet);

extern "C" _declspec(dllexport)
  int _stdcall ClasseDynamique_SetAttribut
    (ClasseDynamique* Objet, int a);

extern "C" _declspec(dllexport)
  int _stdcall ClasseDynamique_GetAttribut(ClasseDynamique* Objet);
```

Définition de la classe

Fonctions manipulant les objets de la classe (en vue de les utiliser en VBA)

Comment utiliser une classe de DLL C++ en VBA (3/5)

- Exemple de **`fichier.cpp`** d'une classe de DLL C++ - en vue d'une utilisation en VBA :

```
ClasseDynamique * _stdcall ClasseDynamique_Constr()
 { return (new ClasseDynamique); } // retourne l'adresse d'un objet
                                   // en mémoire
int _stdcall ClasseDynamique_Destr(ClasseDynamique* Objet)
 { delete Objet; // supprime l'objet en mémoire
   return 1;
 }

int _stdcall ClasseDynamique_SetAttribut
  (ClasseDynamique* Objet, int a)
    { Objet->SetAttribut(a); // Applique la méthode SetAttribut
      return 1;              // à l'objet situé en mémoire
    }

int _stdcall ClasseDynamique_GetAttribut(ClasseDynamique*
Objet) // Applique la méthode GetAttribut à l'objet situé en mémoire
  { return Objet->GetAttribut(); }
```

+ ajouter le corps des méthodes de la classe dans le `.cpp` !! ◁

Comment utiliser une classe de DLL C++ en VBA (4/5)

- Exemple de **`fichier.def`** d'une classe de DLL C++ - en vue d'une utilisation en VBA :

```
LIBRARY MaDll
EXPORTS
   ClasseDynamique_Constr
   ClasseDynamique_Destr
   ClasseDynamique_SetAttribut
   ClasseDynamique_GetAttribut
```

- Exemple de déclaration en VBA :

```
Declare Function ClasseDynamique_Constr Lib "MaDll.dll" () As Long

Declare Function ClasseDynamique_Destr Lib "MaDll.dll" _
  (ByVal Adresse As Long) As Long

Declare Function ClasseDynamique_SetAttribut Lib "MaDll.dll"
(ByVal Adresse As Long, ByVal a As Long) As Long

Declare Function ClasseDynamique_GetAttribut Lib "MaDll.dll" _
(ByVal Adresse As Long) As Long
```

Comment utiliser une classe de DLL C++ en VBA (5/5)

- Exemple de programme VBA :

```
Sub Main()
    Dim Adresse As Long, r As Double
    Dim a As Integer, result As Long
    a = 4
    Adresse = ClasseDynamique_Constr()
    MsgBox ("Nouvel objet instancié à l'adresse : " &
    Adresse)
    result = ClasseDynamique_SetAttribut(Adresse, a)
    MsgBox (a & " a été affecté à l'objet " & Adresse)
    result = ClasseDynamique_GetAttribut(Adresse)
    MsgBox ("L'objet a pour valeur " & result)
End Sub
```

`MsgBox(paramètres)` *affiche une boite de dialogue affichant les paramètres de la fonction*

Complément VBA pour Excel (1/13)

VBA ≠ Excel (fonctions de même nom avec un comportement différent)

- Programmation procédurale et événementielle i.e. : lancement d'un sous programme
  - Par un autre sous-programme
  - A la suite d'un événement (ex. clic)
- Pas de programme principal
- Sites intéressants :
  - `http://ericrenaud.fr/`
  - `http://pagesperso-orange.fr/jml85/Pages/cours_VBA.htm`

Complément VBA pour Excel (2/13)

- Début et fin de programme :

```
Public Sub nom_du_programme()
    ‘ séquences d’instructions
End Sub
```

- 12 types dont les plus utilisés :
  - `String` pour stocker des chaînes de caractères
  - `Integer` pour stocker des valeurs entières
  - `Double` pour stocker des valeurs décimales
  - `Long` pour stocker des grandes valeurs entières
  - `Boolean` pour stocker soit un 0 soit un 1 (un bit)

Complément VBA pour Excel (3/13)

- Correspondance de type C++ / VBA :

Type C/C++	Equivalent Visual Basic
INT	LONG
UINT	LONG
WORD	INTEGER
BYTE	BYTE
CHAR	BYTE
CHAR*	STRING

- Pas d'obligation de déclaration de variables – mais vivement conseillé

```
Dim nom As String
Dim revenu As Long
```

Complément VBA pour Excel (4/13)

- Opérateurs : + - * / etc.

```
Public Sub mon_prog ()
    Dim a As Integer, b As Integer, c As Integer, d As Integer
    a = 5
    b = 7
    c = a + b
    d = a * b
End Sub
```

- Tableaux : deux variantes
  - `Dim tabl1(0 To 12) As Integer`

    13 emplacements pour des entiers

    ```
    tabl1(0) = 14
    tabl1(2) = 17
    ```

  - `Dim tabl2 As Variant`

    + Utilisation de la fonction **Array**

    ```
    tabl2 = Array("Janvier", "Février", "Mars")
    ```

Complément VBA pour Excel (5/13)

- Affichage du contenu d'un tableau :

```
Dim mois As Variant, m As Variant
mois = Array("Janvier", "Mars", "Août", "Décembre")
For Each m In mois
MsgBox m
Next m
```

ou

```
Dim mois As Variant, i As Integer
mois = Array("Janvier", "Mars", "Août", "Décembre")
For i = 0 To 3
MsgBox mois(i)
Next i
```

Complément VBA pour Excel (6/13)

- Fonctions de manipulation de tableaux :
  - **`Lbound`** : plus petit index du tableau
  - **`Ubound`** : plus grand index
  - **`Array(…)`** retourne un tableau (doit être affecté à un **`Variant`**)
  - **`Erase`** efface le tableau de la mémoire
- Instructions conditionnelles :

```
Dim a As Integer, b As Integer
a = 5
If a < 10 Then
  b = 1
Else
  b = 2
End If
```

Complément VBA pour Excel (7/13)

- Boucle **`For`** :

```
Dim a As Integer
For a = val1 To val2
Next a
```

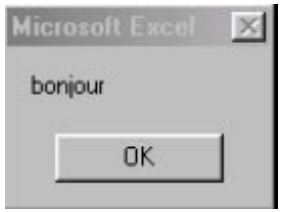
- Boucle **`While`** :

```
Dim a As Integer
a = val1
While a < val2
  …
  a=a+1
Wend
```

Complément VBA pour Excel (8/13)

- Entrées/Sorties :
  - Pas d'exécution dans un terminal
  - Lecture et écriture dans une cellule d'Excel
  - Utilisation de boîtes de Dialogue
  - Exemples :

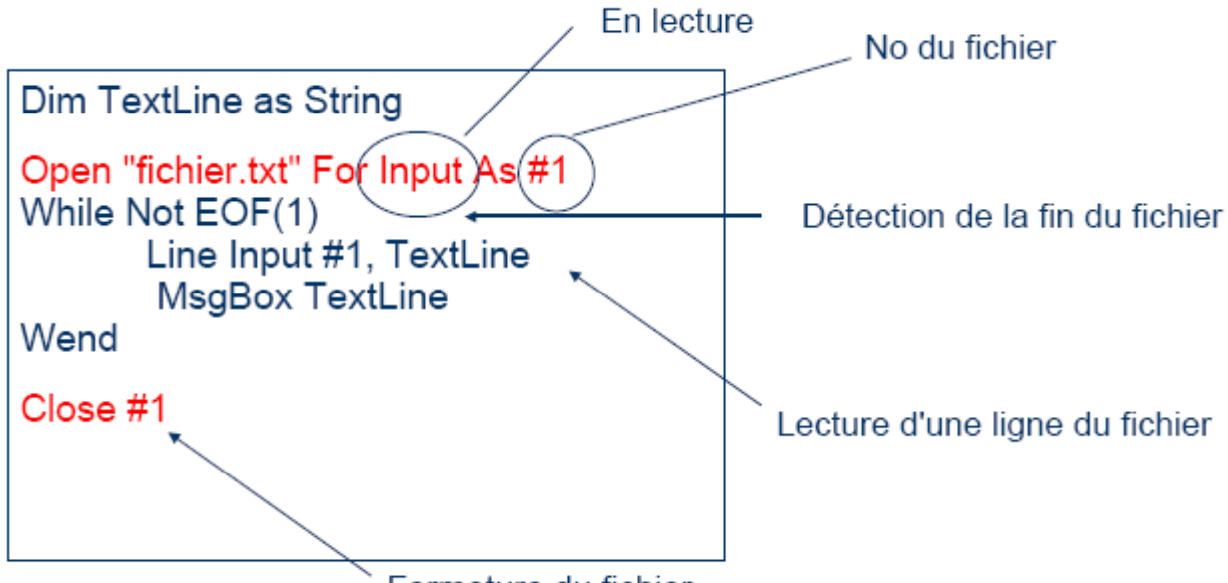
```
msgbox("bonjour")
```




```
Nom = InputBox("votre nom :")
```

Complément VBA pour Excel (9/13)

- Gestion de fichiers – Possibilités de :
  - Créer des fichiers
  - Compléter des fichiers
  - Effacer des fichiers
  - Lire des fichiers
  - Créer et supprimer des répertoires
  - Exemple de lecture :



Complément VBA pour Excel (10/13)

- Fonctions mathématiques :
  - Valeur absolue: **`Abs(-9)`** retourne 9
  - Signe: **`Sgn(-18)`** retourne –1 (ou 0 ou 1)
  - Partie entière:
    `Fix(-18.3) = -18`
    `Fix(18.3) = 18`
    Enlève la partie décimale
  - **`Sqr, Exp, Log`**
    `Sqr(4)` retourne 2, **`Exp(5)`** retourne 148.413…,
    `Log(9)` retourne 2.197224… (en base e)
  - Nombres aléatoires
    `Rnd` retourne un nombre aléatoire entre 0 (compris) et 1 (non compris)
    `a = Rnd`
  - **`Sin, Cos, Tan, Atn`** (arc-tangente)

Complément VBA pour Excel (11/13)

- Définition de fonction :

```
Public Function nom_fonct (variable As
  String) As Integer
    --- instructions
    'valeur de retour:
    nom_fonct=val
End Function
```

- Appel de fonction :

```
b= nom_fonct("test")
call nom_fonct(param)
```

- Passage des paramètres : Par défaut passage par référence

  Pour faire du passage de paramètre par valeur : **`ByVal`**

Complément VBA pour Excel (12/13)

Manipulation des éléments (ou « objets ») d'Excel :

- **`Application`** : contient des **`Workbooks`**
- **`Workbook`** : représente un classeur Excel et contient des **`Worksheets`**
- **`Worksheet`** : représente une feuille de calcul Excel
- **`Cells`** : représente une cellule dont on précise le numéro de ligne et de colonne
- **`Range`** : représente une cellule ou une plage de cellules dont on précise le nom

Voir l'Aide VBA sous Excel!!

Complément VBA pour Excel (13/13)

Exemple de manipulation de cellules :

```
Public Sub Main()
    Randomize 'Initialise le générateur de nombres aléatoires
    For i = 1 To 5
        Cells(i, 1) = Rnd ' Affecte un nombre aléatoire entre 0 et 1
                          ' à la cellule située à la ligne i, colonne 1
    Next i

    Dim r As Double

    For i = 1 To 5
        ' Affecte à r la différence entre la valeur de la cellule C1
        ' et la cellule située à la ligne i, colonne 1
        r = (Range("C1")) - Cells(i, 1)
        msgbox(r) ' affiche la valeur de r dans une boîte de dialogue

    Next i

End Sub
```

Voir l'Aide VBA sous Excel !!