UNIVERSITÉ PARIS DESCARTES

SDA - Structures de Données et Algorithmes

T05 – Commentaires
T06 – Type booléen (bool)
T07 – Déclaration de types
T08 – Principe des entrées-sorties
T10 – Variable et opérateur adresse &
T11 – Pointeur et opérateur d'indirection *
T12 – Référence
T14 – Qualité de l'analyse d'une fonction
T15 – Règle des effets de bord (1)
T16 – Application de la règle au prototypage
T18 – Formalisme de documentation Javadoc
T19 – Documentation de fonction
T20 – Documentation de fichier physique (cartouche)
T22 – Documentation d'algorithme, implémentation

Equipe pédagogique
Marie-José Caraty, Denis Poitrenaud, Julien Rossit, Camille Kurtz, Jacques Alès-Bianchetti, Eloi Keita

Cours n° 1

Analyse, prototypage et documentation des fonctions

Objectifs du cours

- **Modélisation** : le Type Abstrait de Données (TAD)
  modèle algébrique des structures de données et des algorithmes associés
- **Implémentation** d'un TAD
  notion de composant, donnée concrète et structuration de code
- **Etude** des **composants** « techniques » **classiques** en informatique :
  pile, file, liste, arbre, graphe, …
- (**Ré**)**utilisation** des composants « techniques » pour la **résolution de problème**
  (algorithmes de planification, de concurrence,…)
  notion de **généralisation** d'un composant (cf. **généricité** en C++)

Programmation impérative structurée
avec présentation du différentiel des langages C et C++

Pré-requis **Fonctions - Gestion de la mémoire dynamique**

Algorithmique **Algorithmes de tri et de recherche (versions itérative et récursive)**
Notion de complexité d'un algorithme (coûts mémoire et d'exécution)

Organisation du cours

Equipe pédagogique

- Marie-José Caraty
- Julien Rossit
- Camille Kurtz
- Jacques Alès-Bianchetti
- Van Cuong Kieu

Accès aux codes sources des cours et des TD dans le répertoire :
`COMMUN/DUT_1ere_annee/SDA`

Par semaine 1 cours, 1TD et 2TP

Contrôle des connaissances

- **Un projet** **coef. 1.5**
- **Un DST la semaine du 16 janvier 2017** **coef. 3**

Environnement de programmation

- **Visual Studio 2017 : EDI (Environnement de Développement Intégré)**
  propriété de Microsoft (**VB, C, C++, C#,**…)

Sommaire

Commentaires

C++

Le commentaire précède immédiatement ce qu'il commente (sans ligne blanche)

- `/* en C et en C++, commentaire classique sur plusieurs lignes */`
- `a = 2 * b; // En C++, commentaire jusqu'au saut de ligne`
- Commentaire étiqueté utile à documenter une portion de code et à imbriquer des commentaires

```
/* COM1 Commentaire principal */
inst_1;
inst_2;
/* COM2 Commentaire imbriqué */
inst_10;
inst_11;
/* FIN_COM2 */
inst3_3;
/* FIN_COM1 */
```


Type booléen (bool)

C++

Type booléen natif en C++ `bool`

- Deux valeurs possibles : `false` et `true` de valeurs respectives `0` et `1`
- Toute expression relationnelle renvoie une valeur booléenne de type `bool`

Exemple : `(i==0)` a pour valeur `true` si `i` vaut `0`, `false` sinon

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    cout << "bool : " << sizeof(bool) << " octet"
         << " - Domaine : " << false << " .. " << true << endl;
    cout << "bool : " << sizeof(bool) << " octet"
         << " - Domaine : " << boolalpha
         << false << " .. " << true << endl;
    system("pause"); return 0;
}
```

```
bool : 1 octet - Domaine : 0 .. 1
bool : 1 octet - Domaine : false .. true
```


Déclaration de types

C++

Déclaration de types

typedef n'est plus utile à la déclaration des types énumérés ou structurés

```
enum Sexe{MASCULIN, FEMININ};  // Sexe est un type énuméré

typedef char Ch20[21];      // CH20 est le type d'une chaine de caractères
    //  les caractères « utiles » sont stockés de 0 à 19  (20 caractères utiles)
    //  se terminant par '\0'  à l'index 20

struct Personne {           // Personne est un type enregistrement/utilisateur
   Ch20 nom;
   int age;
   Sexe sexe;
   bool marie;
 };

 Personne p;
 p.sexe=FEMININ; p.marie=true;  // accès aux attributs par la notation pointée
```

les champs du type sont également appelés des « attributs »

Principe des entrées-sorties

C++

Inclusion et clause d'utilisation

`#include <iostream>`	fichier d'entête où sont déclarées les opérations sur les flots d'entrée/sortie (E/S)
`using namespace std;`	clause d'utilisation de la bibliothèque standard `std`, sans cette clause, préciser le chemin de la bibliothèque standard (std::)
`cin et cout`	les flot d'E/S standards (clavier et écran)
`>>`	l'opérateur binaire de lecture/d'extraction de flot
`<<`	l'opérateur binaire d'écriture/d'insertion dans un flot
`endl ou '\n'`	saut de ligne

```
// Lecture d'un entier et d'un caractère
int i; char c;
cout << "Entier (i) et caractère (c) ? " << endl;
cin >> i >> c;
cout << "i=" << i << " et c=" << c << endl;
```

Les entrées-sorties

```
Entier (i) et caractere (c) ?
69 W
i=69 et c=W
```

C

```c
// main01.c
#include <stdio.h>
#include <stdlib.h>
int main() {
   int i; char c;
   printf("Entier (i) et caractere (c) ? \n");
   scanf("%d %c", &i, &c);
   printf("i=%d et c=%c \n",i, c);
   system("pause"); return 0;
}
```

Plus de format!

C++

```cpp
// main01.cpp
#include <iostream>
using namespace std;
int main() {
   int i; char c;
   cout << "Entier (i) et caractere (c) ? " << endl;
   cin >> i >> c;
   cout << "i=" << i << " et c=" << c << endl;
   system("pause"); return 0;
}
```

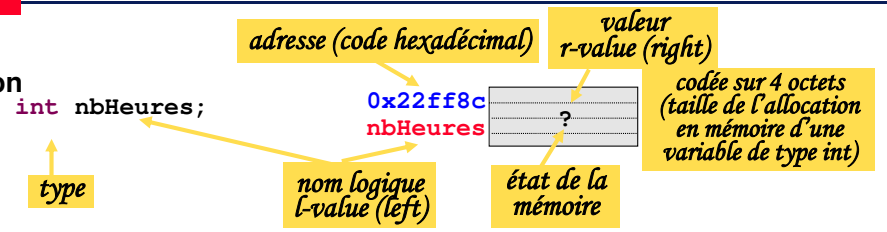

littéraux (*constantes de type*) et *expressions* sont des *r-values*
Ex: `0, 2.7, 'z', "Hello", 365/3%30, 2*nbJours`

Variable et opérateur de référencement (&)

C C++

Rappel

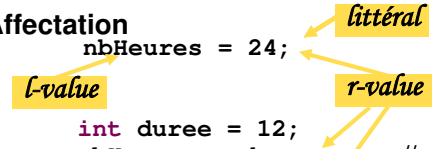
Déclaration



Une variable (**nbHeures**) est un nom logique (**nbHeures**) pour une adresse mémoire (**0x22ff8c**) où sa valeur (? indéfinie à ce niveau de déclaration) est stockée sur la taille mémoire (4 octets) allouée pour un codage binaire associé à son type (**int**)

- Accès à la **valeur** (**r-value**) de la variable par son **nom logique nbHeures**
- Accès à **l'adresse-mémoire** de la variable par l'**opérateur de référencement & : &nbHeures**

Affectation



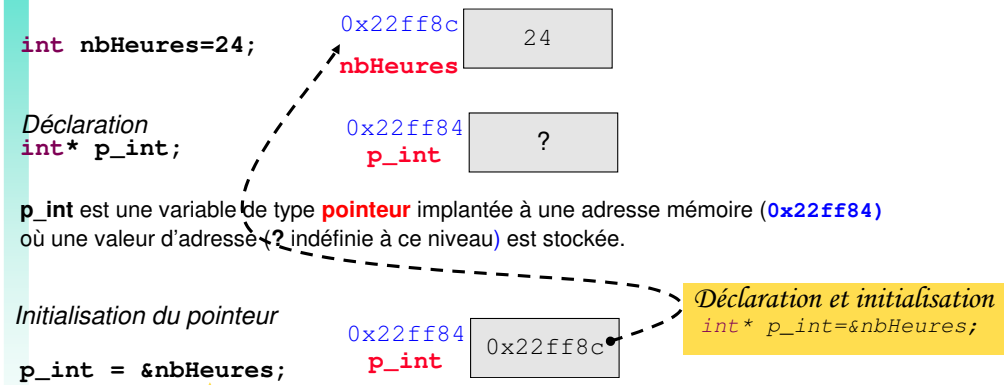

```
int duree = 12;
nbHeures = duree;     // affectation à partir de la r-value / valeur (12) de duree
nbHeure = ++nbHeure;  // affectation à partir de l'incrément de la r-value de nbHeure
2*Duree = nbHeures;   // Erreur, Mess. Comp. : « non-lvalue in assignment »
```


Déclaration et initialisation de pointeur
`[const]<id_type>* <id_variable>[=&<id_variable>];`

Pointeur et opérateur de référencement

C C++

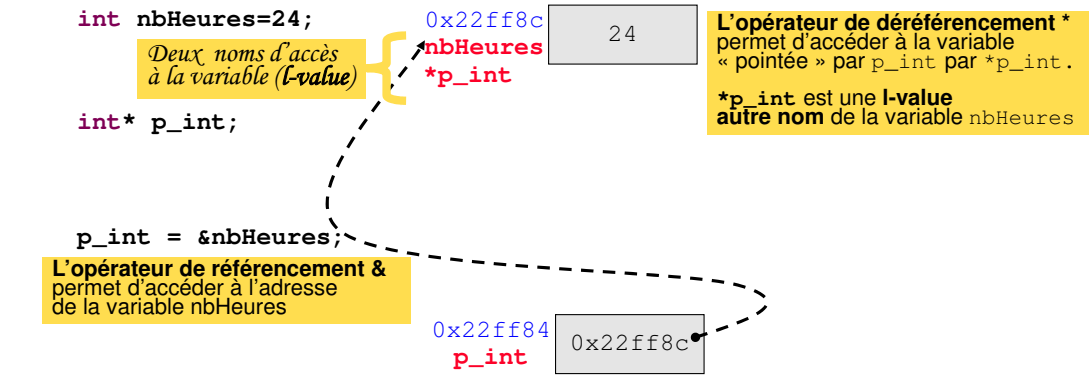
Rappel



Pointeur et opérateur de déréférencement ou d'indirection (*)

C C++

Rappel




```
cout << p_int;      // affiche 0x22ff8C
cout << &p_int;     // affiche 0x22ff84
cout << *p_int;     // affiche 24 (r-value)
cout << *p_int+2;   // affiche 26 (calcul avec la r-value de *p_int)
```

Référence

C++

Déclaration et initialisation de référence
[const]<id_type>& <id_variable>=<id_variable>;

```
int nbHeures = 24;
```

0x22ff8c nbHeures | 24

Déclaration et initialisation obligatoire

la référence — *la variable référencée*

```
int& r_int = nbHeures;
```

0x22ff8c nbHeures, r_int | 24

Deux noms logiques pour la même variable implantée en mémoire
r_int est un alias de nbHeures

Impossible de déclarer une référence sans l'initialiser

Une fois initialisée, une référence ne peut plus référencer une autre variable

```
int duree = r_int; // duree vaut 24
```

0x22ff90 duree | 24

```
nbHeures = 12;      // r_int vaut 12
r_int++;            // r_int et nbHeures vallent 13
r_int = duree;      // r_int et nbHeures vallent 24
```

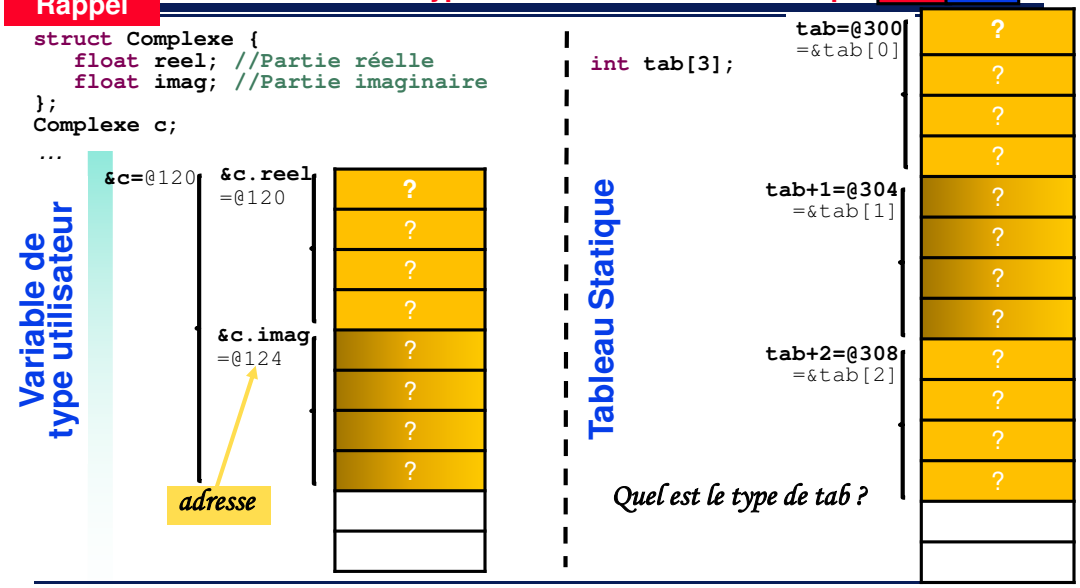
Déférencement inutile contrairement au pointeur

Représentation en mémoire – Variable de type-utilisateur et tableau statique

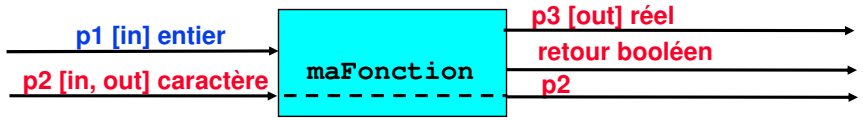
C C++

Rappel

```
struct Complexe {
   float reel; //Partie réelle
   float imag; //Partie imaginaire
};
Complexe c;
...
```



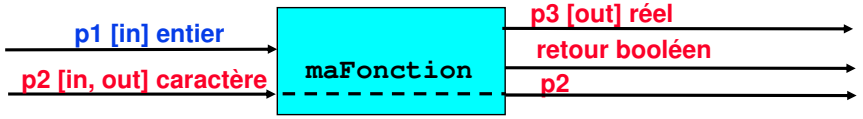
Les 4 types de paramètres d'une fonction (1/2)



[in] Paramètre d'entrée
paramètre utile « au calcul » du/des résultat(s) de la fonction
Caractéristique : se trouve en partie droite d'une affectation dans le corps de la fonction

[out] Paramètre de sortie
Paramètre « porteur » d'un résultat
Caractéristique : se trouve en partie gauche d'une affectation

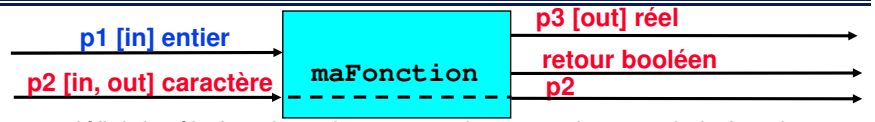
Les 4 types de paramètres d'une fonction (2/2)



[in, out] **Paramètre d'entrée/sortie**
paramètre utile « au calcul » du/des résultat(s) de la fonction mais aussi paramètre porteur d'un résultat
Caractéristique: se trouve en partie droite d'une affectation dans le corps de la fonction mais également en partie gauche d'une affectation

retour **Retour**
Résultat de type scalaire (affectable) renvoyé en fin du corps de la fonction signalé par le mot clé *return* suivi de la variable ou expression du résultat

Qualité de l'analyse d'une fonction



Rôle définir le rôle fonctionnel correspondant au traitement de la fonction

Interface définir l'ensemble de ses paramètres formels] et/ou un retour
- les entrées nécessaires au traitement [in]
- la (les) sortie(s) (1) résultat(s) calculé(s) lors du traitement et (2) à transmettre à l'appelant via les paramètres formels [out] et [in,out] ou un return (d'un résultat scalaire)

Qualité du nommage des entités (variables, fonctions, types, …)

Qualité du choix du prototype parmi tous les prototypes possibles
- ergonomie de l'appel
- optimisation du temps d'exécution à l'appel de la fonction (temps de recopie des paramètres effectifs dans les paramètres formels)

Qualité du choix du type
typer les paramètres au plus proche des données
- **éviter des préconditions**
- optimiser la place mémoire

Connaissances acquises en IAP

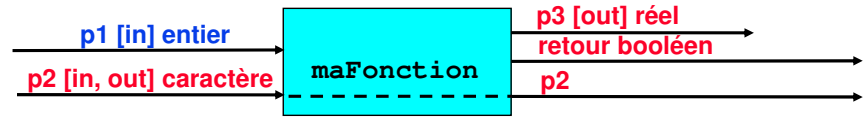
Règle n° 1. des effets de bord – Prototypage des paramètres d'entrée et de sortie

R1.		Nature du paramètre	Prototype	Nature de l'appel `pE` : paramètre effectif de type `T`
Paramètre d'entrée p [in] T	**Paramètre formel p de type T (tableaux exclus)**			
	C++ C	variable	`void f(T p);`	`f(pE);` (variable) `f(2.0);` (littéral) `f(2.*pE);` (expression) *T réel*
Paramètre de sortie p [out] T *ou* p [in,out] T	**Paramètre formel p de type T (tableaux exclus)**			
	[out]/[in,out] C++	référence	`?`	`?`
	[out]/[in,out] C++ C	pointeur	`void f(T* p);`	`f(&pE);`
	Paramètre de retour			
	[out] C++ C	paramètre de retour	`T f();`	`T res; res=f();`

Remarque : dans certains langages (Ada, Eiffel), les paramètres [out] et [in,out] sont distingués par leur syntaxe.
Le compilateur vérifie la cohérence du code relativement à la nature des paramètres (violation, inutilisation)

Application de la règle n°1 au prototypage

Analyse



Déduction du prototype

```
bool  maFonction(unsigned int p1, char* p2, float* p3);
```


Documentation de code et génération de document

Documents de la production logicielle

Parmi tous les documents à produire en fin de projet (industriels/académique), plusieurs documents techniques (livrables) sont liés au code-source :
la production brute en informatique

Utilité de la documentation technique

Documentation livrée au « client » pour la maintenance (évolutive ou adaptative) des produits logiciels
Capitalisation des connaissances : réutilisation (de tout ou partie) des produits logiciels développés

Intérêt

Produire automatiquement cette documentation à partir des commentaires du code-source

Principe

Utiliser des étiquettes sémantiques (*tags*) incluses dans les commentaires du code qui permettent d'extraire l'information « pertinente »
Exemples d'utilitaires : Javadoc, Doxygen, …
Formats usuels : HTML (documentation on-line hypertexte) [cf. bibliothèques]
rtf (Rich Text Format), Word, Latex… [cf. livrables de projets]

Tags interprétés par Javadoc et Doxygen

Spécifiques aux unités de compilation (fichiers physiques)

`@author` identification de l'auteur
`@version` version (n°, date de création/modif.)

`@file` nom du fichier physique
`@brief` rôle de l'unité de compilation

Spécifiques aux fonctions

`@brief` rôle de la fonction

`@param` nom et description de l'argument
`@return` description du résultat de retour
`@exception` nom et description de l'exception lancée

`@param[in]` paramètre d'entrée
`@param[out]` paramètre de sortie
`@param[in,out]` entrée-sortie

`@pre` préconditions
`@post` postconditions

Toute entité

`@see` identification de lien avec d'autres entités
`@deprecated` indique un élément obsolète, la raison de l'obsolescence et ce qu'il faut utiliser à la place
`@since` version à laquelle l'élément a été ajouté

Tags additionnels de Doxygen

Documentation de fonction

- **Déclaration de commentaire** /** … */ et libellés des **tags** sont **imposés**
- Présentation libre à l'intérieur du commentaire « Javadoc » (/** … */)

Documentation associée au prototype de la fonction ou à la définition de la fonction

```
/* @brief Addition de deux horaires
 * @param[in] h1 le premier horaire
 * @param[in] h2 le deuxième horaire
 * @param[out] res addition de h1 et h2
 * @pre h1 et h2 valides
 */
void addition(const Heure* h1, const Heure* h2, Heure* res);
```

Documentation de la fonction `(addition)`

Remarque
- Autant de `@param` que de paramètres formels, donnés dans l'ordre des déclarations des paramètres
- @ return donné en dernier (en cas de paramètre de retour)

Documentation de fichier physique – Cartouche

Déclaration de commentaire /** … */ et libellés des **tags** sont **imposés**
Présentation libre à l'intérieur du commentaire « Javadoc » (/** … */)

Exemple de documentation

```
/**
 * @file gestionHoraire.cpp
 * @brief Utilitaires de gestion d'horaires
 * @author l'équipe pédagogique
 * @version 1.0  08/11/2016
 */
```

Cartouche: documentation du fichier physique `GestionHoraire.cpp`

Tout fichier physique doit avoir un cartouche

`.cpp` programme principal ou corps de composant (Notion différée)
`.h` entête (header) de composant (Notion différée)

Documentation de blocs – Objectifs et usages

Bloc : {<`séquence d'instructions`>} (ex : un corps de fonction)
A ce niveau, les commentaires sont usuellement destinés à la maintenance
utiliser `/* ...*/` ou `//`
En cas de génération automatique, utiliser le formalisme Javadoc `/** … */`

Objectifs
- Commenter les grandes étapes d'un l'algorithme (rôle fonctionnel du traitement)
  Le commentaire porte sur une instruction composée ou sur une séquence d'instructions (niveau macroscopique)
- Variables (à l'exception des variables de boucles) et types sont à commenter
- Commenter les singularités de code

La forme
- Une ligne de code ne doit pas excéder 80 colonnes pour la lisibilité des listings (conservation de l'indentation)
- Le commentaire précède juste ce qu'il commente (sans ajout de ligne blanche)
- Le commentaire devient gênant lorsqu'il masque l'indentation du code
  Indenter le commentaire sur le code qu'il concerne
- Une façon de commenter une séquence d'instructions est d'attacher une étiquette en début de commentaire et d'indiquer sa fin par cette même étiquette (permet l'imbrication de commentaires)

Exemple de documentation d'un algorithme

Problème traité : trouver la liste des nombres premiers dans l'intervalle [1, n]

Algorithme du crible d'Eratosthène

1) Construire le crible : une liste des entiers naturels de 2 à n

2) Supprimer du crible tous les entiers qui ne sont pas premiers :
Balayer le crible depuis son début jusqu'à sa fin*
si un nombre est trouvé dans le crible (il est premier) alors
supprimer du crible tous les multiples de ce nombre

En fin de traitement, le crible contient les nombres premiers dans l'intervalle [1, n]

Remarque* : *on peut démontrer que l'on peut limiter le traitement aux nombres inférieurs ou égaux à racine de n (limite du balayage à introduire)*

Application : n=30

Crible

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Liste des nombres premiers de 1 à 30 : 2 3 5 7 11 13 17 19 23 29

En bleu, les choix (e.g., structures de données) etcommentaires lés à l'implémentation.

Algorithme et implémentation

Problème traité : trouver la liste des nombres premiers dans l'intervalle [1, n]
Algorithme du crible d'Eratosthène

1) Construire le crible : une liste des entiers naturels de 2 à n
Choix de la structure de données (plusieurs choix possibles)
on choisit de représenter le crible (la liste) par un tableau de n éléments (indicés de 0 à n-1) représentant la liste des entiers (de 1 à n) Rem : 1 n'est pas premier
Construire le crible : initialiser le tableau (traiter 1)

2) Supprimer du crible tous les entiers qui ne sont pas premiers :
Balayer le crible depuis son début jusqu'à sa fin*
Balayer le tableau (de l'indice 0 ou 1 à LimiteRecherche)
si un nombre est trouvé dans le crible (il est premier) alors
la première occurrence (au poste i) de nombre non nul dans le tableau correspond à un nombre premier
supprimer du crible tous les multiples de ce nombre
technique de marquage : marquer tous les multiples de i+1 comme non premier en mettant l'élément à 0

En fin de traitement, le crible contient les nombres premiers dans l'intervalle [1, n]
Tous les éléments non nuls du tableau sont des nombres premiers

Remarque* : *on peut démontrer que l'on peut limiter le traitement aux nombres inférieurs ou égaux à racine de n (limite du balayage à introduire)*
La limite du balayage est une constante (LimiteRecherche)

Source et commentaires (de l'algorithme) (1/2)

```
// main04.c
#include <stdio.h>
#include <math.h>
int main() {
  /* Com1. Caractéristiques du crible */
  const int N = 200;          // Taille du crible
  const int nb=15;            // Format d'édition
  int crible[N];              // Crible d'Eratosthène
  const int limiteRecherche=(int)sqrt((double)N); // Limite du balayage
  /* fin Com1. */
  int i, j;

  /* Com2. Implémentation de l'algorithme du crible d'Eratosthène */
  /* Initialisation du crible */
  for( i=0; i<N; ++i )
    crible[i] = i+1;

  /* Com3. Suppression des nombres non premiers du crible */
  crible[0] = 0;              // 1 n'est pas premier
  /* Balayage du crible */
  for( i=1; i<limiteRecherche; ++i )
    if( crible[i] != 0 )   // (i+1) est premier
      /* Suppression du crible des multiples (non premiers) de (i+1) */
      for( j=i+1; j<N; ++j )
        if( crible[j] != 0
            && crible[j]%crible[i] == 0 ) //(j+1) est multiple de (i+1)
            crible[j] = 0; // Suppression de (j+1) du crible
  /* fin Com3. */
  /* fin Com2.*/
```

étiquette (label) de début de commentaire

étiquette de fin de commentaire

Source et commentaires (de l'algorithme) (2/2)

```
  /* Com4. Impression des nombres premiers (nb par lignes) */
  printf( "Ensemble des nombres premiers entre 1 et %d :\n", N );
  for( i=1; i<N; ++i ) {
    static int nbParLigne = 0;
    if ( crible[i] != 0 ) {
      if( (nbParLigne++)% nb == 0 ) printf( "\n" );
      printf( "%4d", crible[i] );
    }
  }
  /* fin Com4. */
  return 0;
}
```

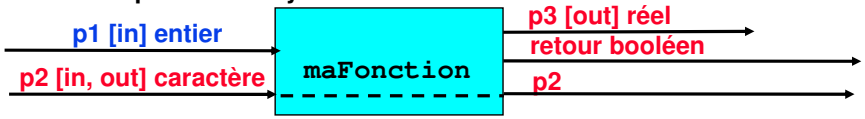
```
Ensemble des nombres premiers entre 1 et 200 :

   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47
  53  59  61  67  71  73  79  83  89  97 101 103 107 109 113
 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
 199
```

Ce que vous avez appris aujourd'hui…

Des spécificités du C++ relativement au type booléen, aux déclarations des types (énumérés et structurés) et aux entrées-sorties

Les objets et opérateurs : variables et opérateur de référencement &, pointeur et opérateur de déréférencement (ou indirection) *, la déclaration et initialisation d'une référence

Retour sur la **qualité d'analyse d'une fonction**



La **règle des effets de bords** qui permet de déduire le prototypage d'une fonction

La **documentation** du logiciel avec le formalisme « Javadoc »
Documentation des **fonctions** (informations à renseigner obligatoirement)
et d'un **fichier physique** appelée **cartouche** (informations à renseigner).

Associée à Doxygen, permet une génération automatique de documentation

La semaine prochaine : la fin du cours i) sur les fonctions (pointeurs et références) …