Formation C Initiation

Gilles Maire

2017

Plan de la formation

Introduction

Rubriques

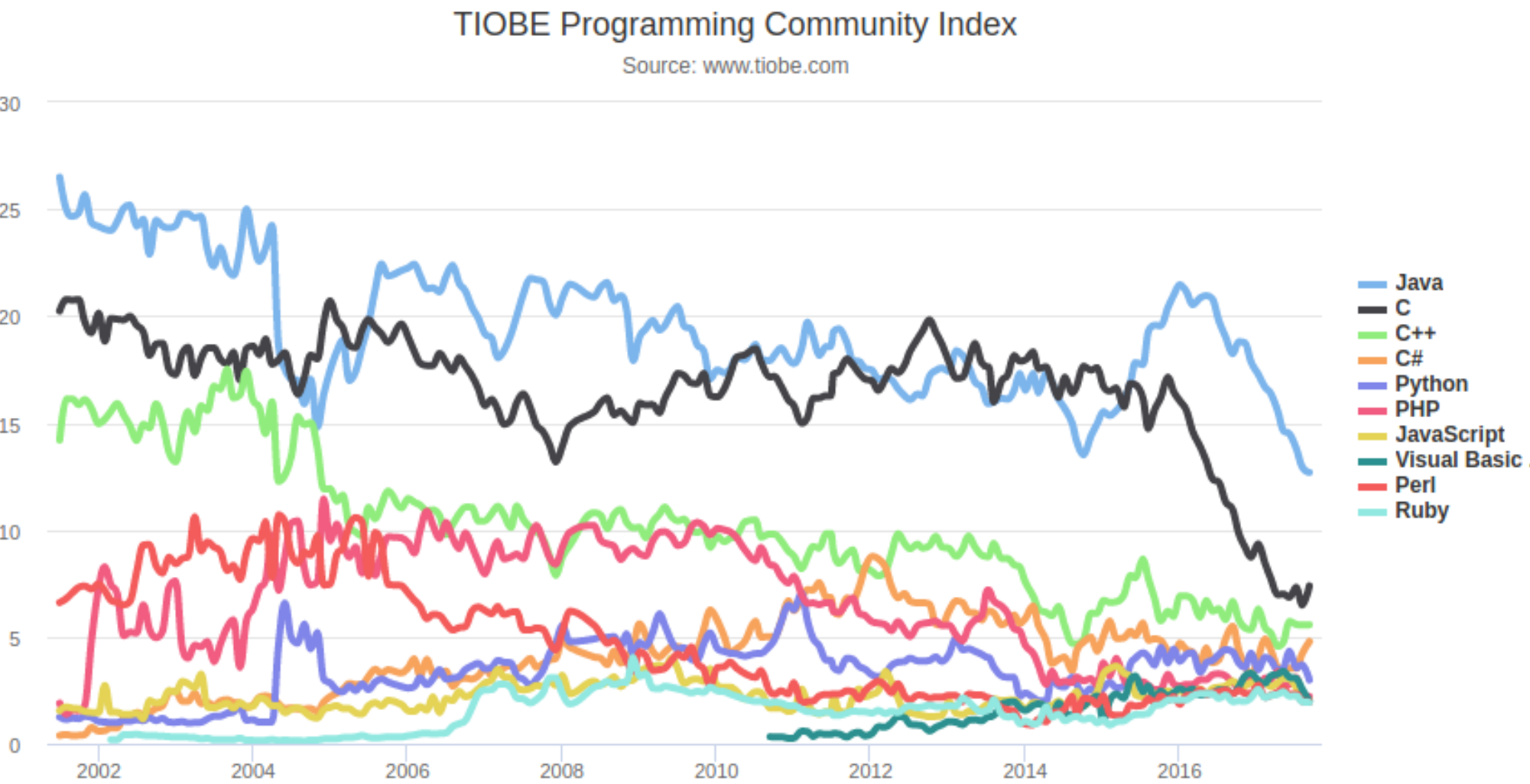
- Introduction
- Outils de développement
- Premier programme

Introduction

Présentation du langage C

- C est un langage de bas niveau dans la mesure où il n'est pas très différent des instructions générées par un macro assembleur. En d'autres termes lorsqu'il est sorti en 1972, les personnes qui programmaient en assembleur l'on très vite adopté et il est toujours couramment utilisé
- Ses inventeurs furent Dennis Ritchie († 12 oct 2011) et Ken Thompson et il vit le jour en même temps qu'Unix dans les laboratoires de Bell.
- **Avantages :** ouvert, standardisé, théoriquement portable, simple
- **Inconvénients :** pas objet, pas de gestion des exceptions, pièges de portabilité, gestion mémoire primitive et source de bug ou de faille de sécurité.

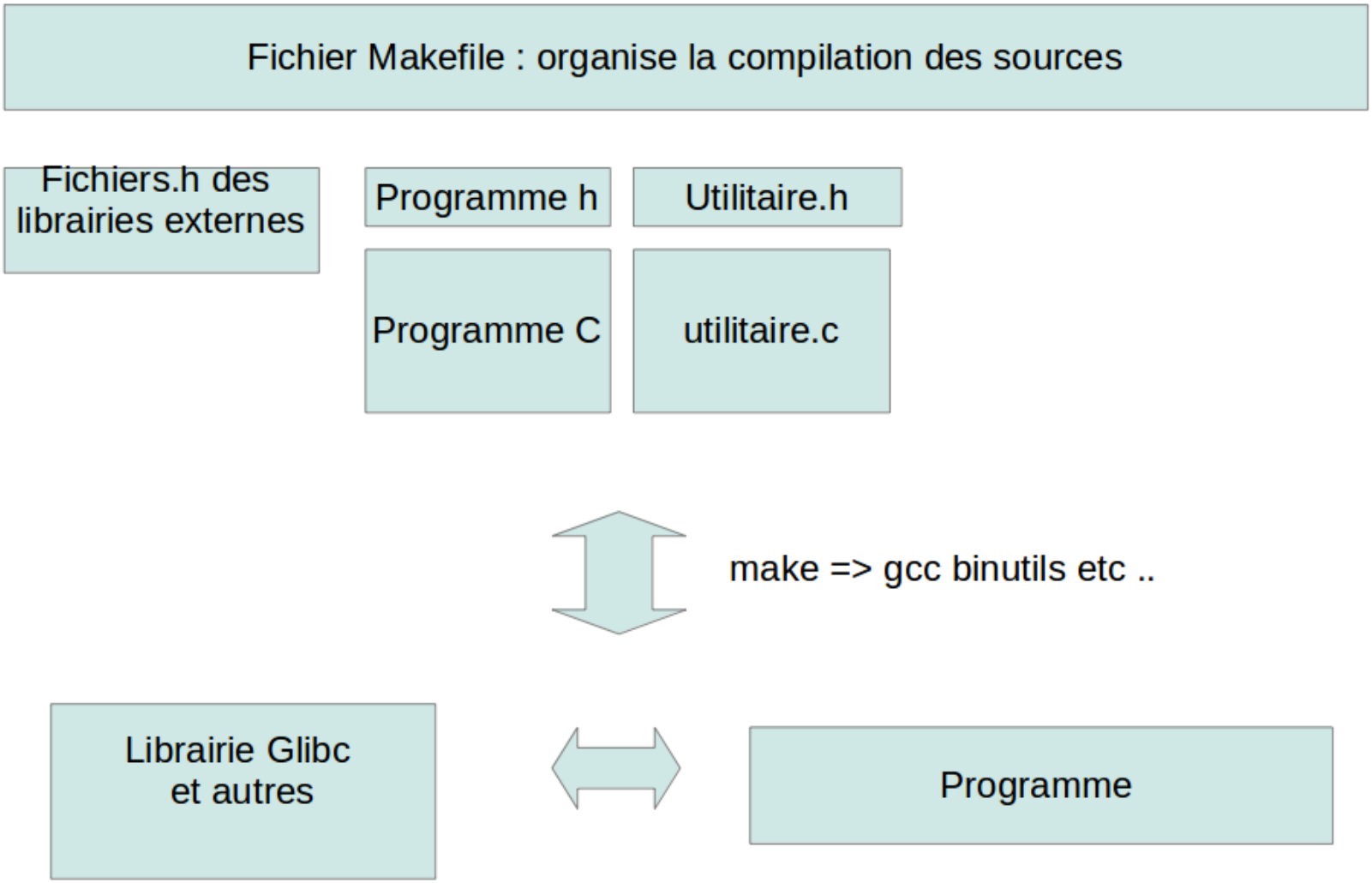
Évolution du C et du C++



Principe de compilation

- La compilation est la transformation d'un programme écrit dans un langage vers un exécutable destiné à un processeur choisi
- On peut compiler sur un équipement un code C pour un processeur qui n'est pas celui de l'ordinateur qui compile, on parle alors de compilation croisée
- L'exécutable fait appel à des librairies qui devront être présentes sur le système
- Un fichier Makefile organise la compilation et définit les différentes tâches

Compilation des programmes



Écriture d'un programme

- Un programme C ou C++ est écrit en langage humain
- On l'édite en format ASCII dans un éditeur de texte qui est plus adapté que Word ou Open Office
- On peut également l'éditer dans un éditeur de haut niveau qui lance de façon transparente la compilation, l'exécution et le débogage (c'est à dire l'aide à la résolution de problèmes)
- Mais on peut aussi lancer à la main la compilation dans une console.

Outils de développement

Les compilateurs C

Il existe plusieurs familles de compilateur

- Sous Windows : Visual C/C++ Borland C/C++ MinGW (à base de GNU)
- Sous MacOSX : XCode
- Sous Linux : GNU CC (support de nombreux processeurs)

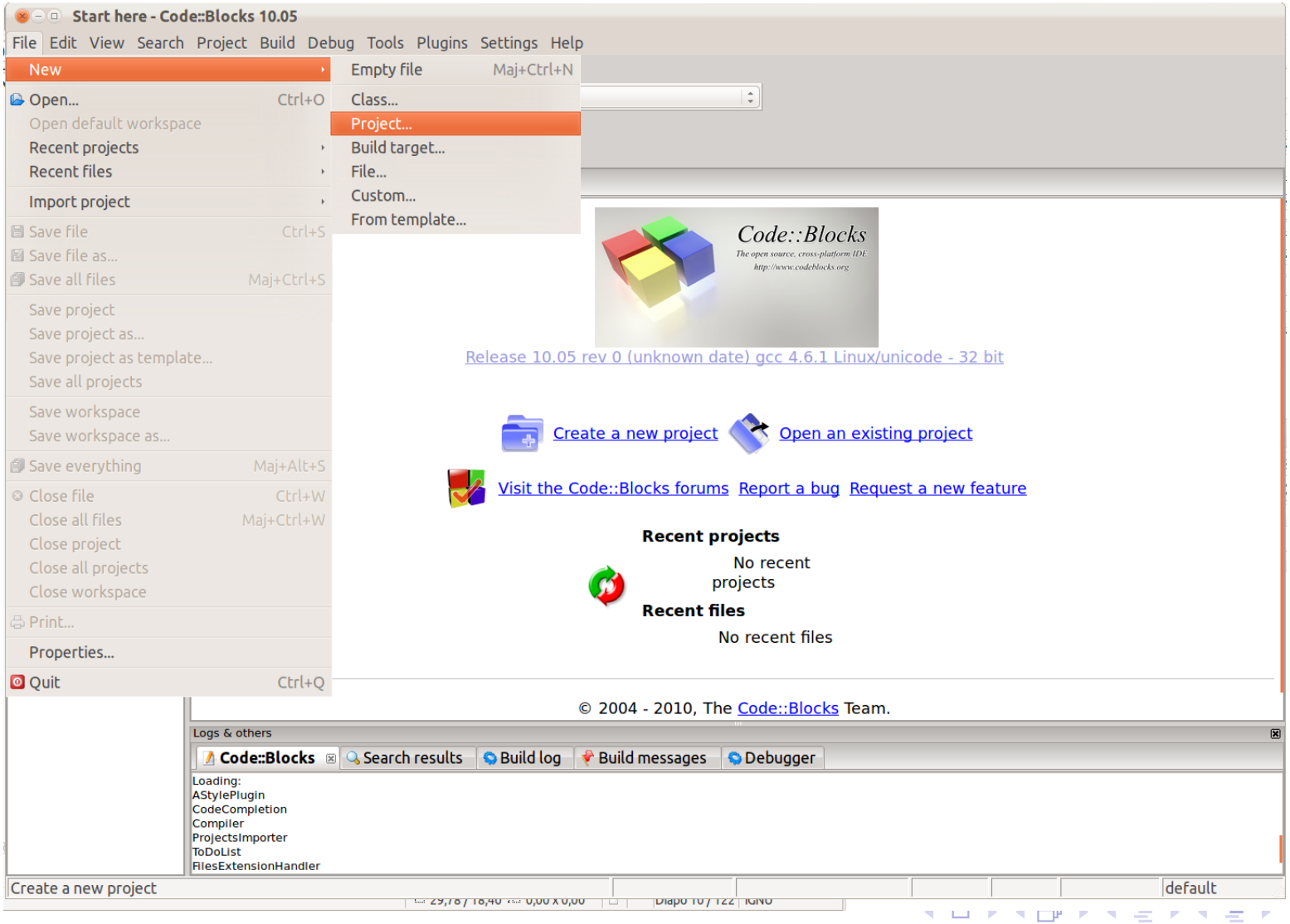
Environnements de développement

- Il existe beaucoup d'environnements de développement, ceux des société Borland, Microsoft, le produit Eclipse d'Oracle, QtCreator de Qt. On trouve beaucoup de produits Open Source qui tournent sur MacOSX Windows et Unix ainsi que sur le compilateur open source GCC
- Nous allons utiliser l'un d'entre eux Code::Blocks.
- Avant d'apprendre C nous allons apprendre à utiliser Code::blocks dans les axes suivants :
  - éditeur de code ;
  - lancement de la compilation
  - lancement du programme.

Installation de Code::Block

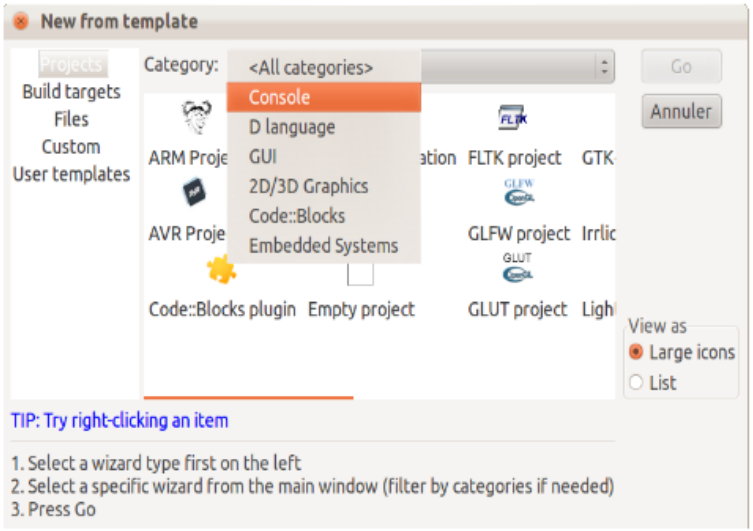
- Allez sur le site http://www.codeblocks.org/
- Dans la partie Download/Binaries on charge
  - codeblocks->version>mingw-setup.exe
- Codeblock est livré avec
  - l'environnement lui même
  - un compilateur C et un compilateur C++ qui sont nécessaires pour compiler les programmes C et C++ c'est mingw

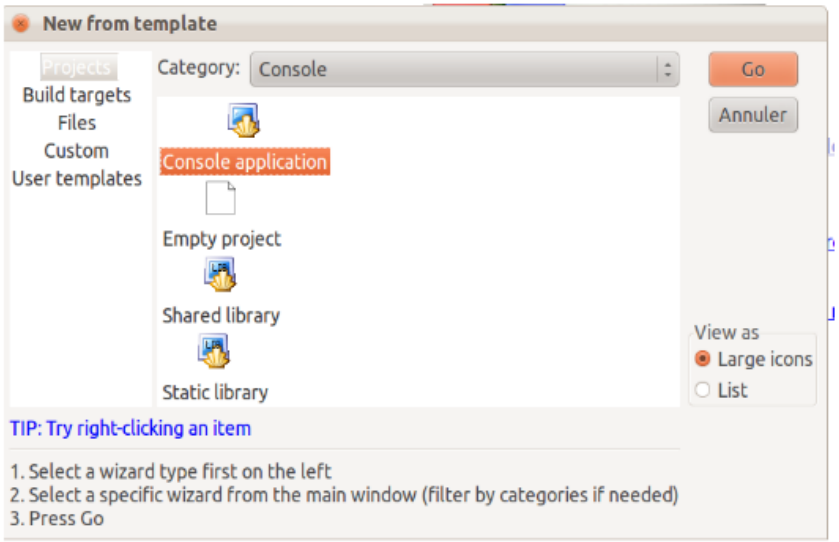
Création d'un nouveau projet

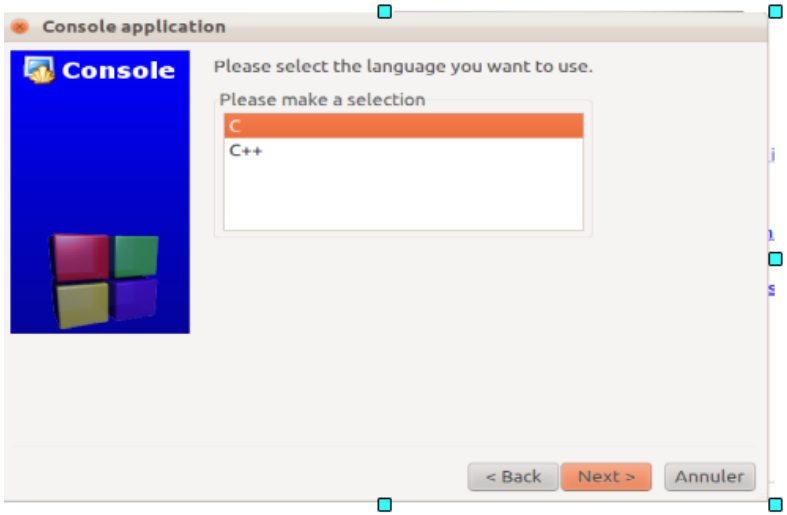


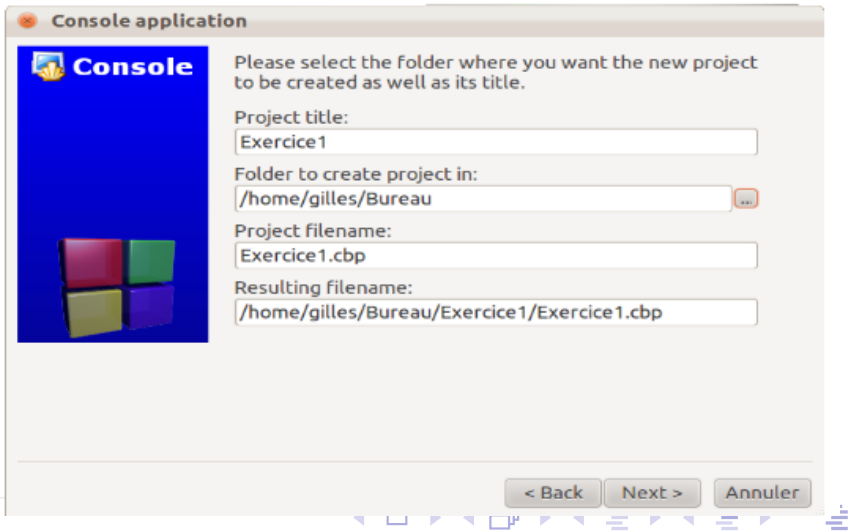
Premier programme

Ouvrir un projet C

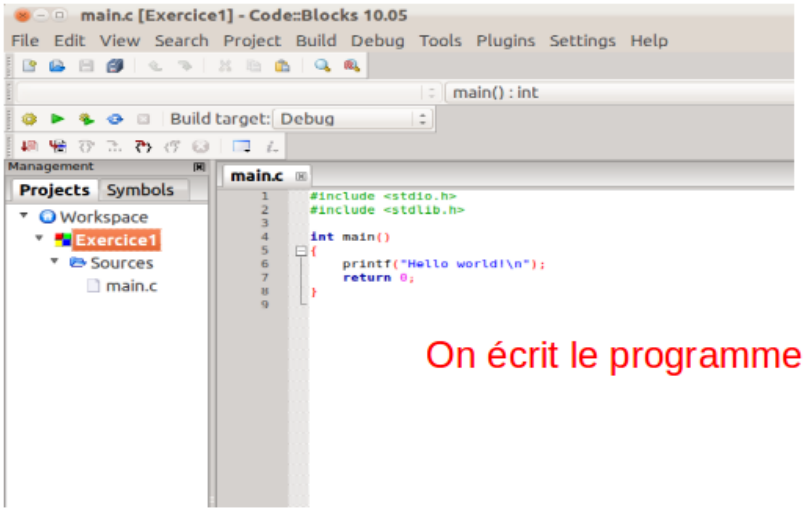




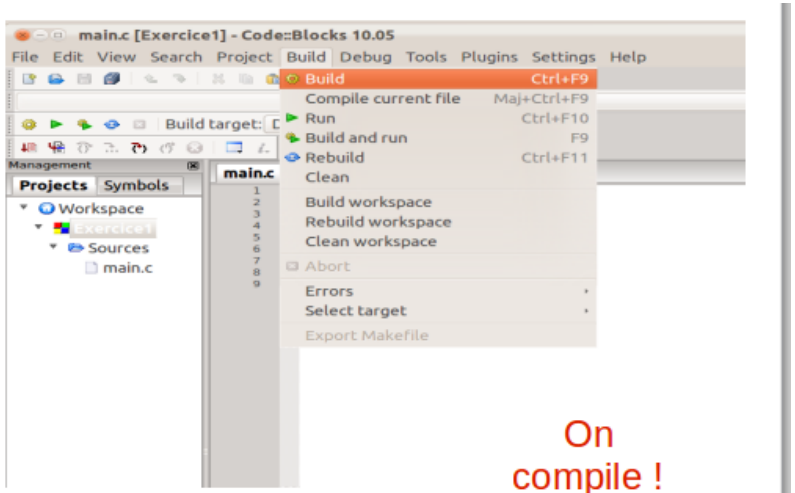




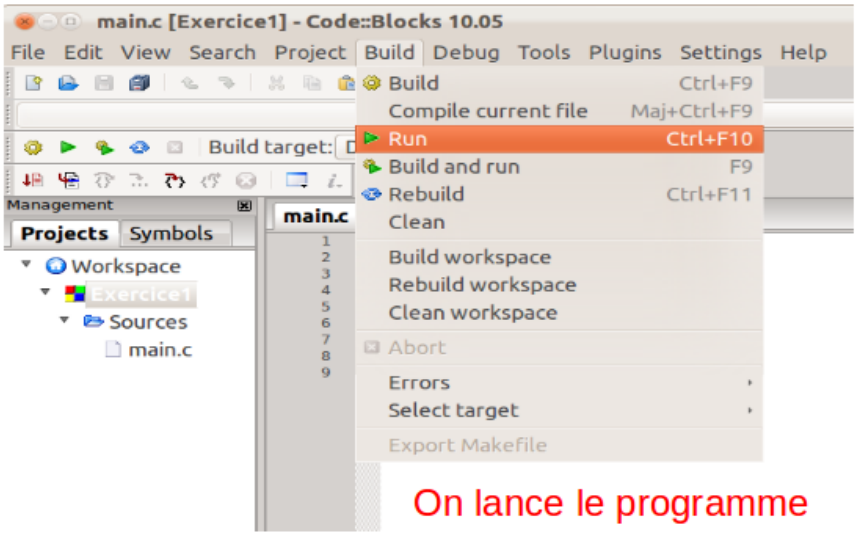
Le premier programme



On écrit le programme



On compile !



On lance le programme

Le programme s'exécute

Source du premier programme

```c
#include <stdio.h>

void main()
{
    printf("\nExemple de programme c\n");
}
```

Eléments

Rubriques

- Vocabulaire
- Blocs d'instruction
- Déclarations

Vocabulaire

Caractères autorisés

- Lettres majuscules
- Lettres minuscules sans accent
- Chiffres
- Caractères de soulignement _
- Caractère d'espace, de tabulation, de retour arrière, de nouvelle ligne, de saut de page
- Caractères spéciaux et de ponctuation `: , ! | ; / : \ ? ~ ' + " # ( % ) & [ ^ ] * { - } = < >` qui ont tous une signification particulière

Les identificateurs

- L'identificateur donne un nom à une entité du programme qui peut être :
  - un nom de variable ou de fonction,
  - un type défini par `typedef`, `struct`, `union` ou `enum`,
- Un identificateur est une suite de caractères parmi :
  - les lettres (minuscules ou majuscules qui sont des lettres différentes, mais non accentuées),
  - les chiffres mais pas en première position de l'identificateur.
  - le "blanc souligné" (_).
- Le compilateur peut tronquer les identificateurs au-delà d'une certaine longueur. Cette limite dépend des implémentations, mais elle est toujours supérieure à 31 caractères.

Séparateurs

- un ou plusieurs espaces blancs les retours à la lignes sont des séparateurs
- ils ont la même fonction qu'une ou plusieurs tabulations
- les instructions sont séparées par des ;
- on peut mettre plusieurs instructions séparées par des ; sur une même ligne.
- les directives de compilation ne sont pas séparées par des ;

Blocs d'instruction

Les mots-clefs

- les spécificateurs de stockage : `auto` `register` `static` `extern` `typedef`
- les spécificateurs de type : `char` `double` `enum` `float` `int` `long` `short` `signed` `struct` `union` `unsigned` `void`
- les qualificateurs de type : `const` `volatile`
- les instructions de contrôle : `break` `case` `continue` `default` `do` `else` `for` `goto` `if` `switch` `while`
- divers : `return` `sizeof`

Blocs d'instructions

- Un bloc est une suite d'instructions élémentaires délimitées par des accolades.
- Un programme C peut contenir un nombre quelconque de blocs.
- Ces blocs peuvent être inclus les uns dans les autres.
- Un bloc peut contenir des déclarations et des définitions.
- Les déclarations de variables à l'intérieur d'un bloc ne sont définies que dans ce bloc
- On peut définir un bloc à tout moment pour y déclarer des variables qui auront une durée de vie limitée au bloc

Les commentaires

- Le C donne la possibilité de placer des explications dans le texte d'un programme. Cela se fait au moyen de commentaires.
- Un commentaire est une suite de caractères placés entre les délimiteurs : /* et */
- **Exemple :**

```
/* ceci est un commentaire à destination des codeurs */
```

- Dans certains cas on trouve des commentaires sous la forme /** */ qui sont toujours des commentaires mais qui sont lus par des programmes de documentation pour générer la documentation de programmation - voir `Doxygen`

Programme main

- C'est une fonction comme nous les verrons plus loin, elle suit la structure :

```
void main(){
déclarations;
instructions;
}
```

- Le mot `void` indique que la fonction ne retourne pas de valeur mais elle peut retourner un code d'exécution
- Nous verrons ce point dans les fonctions
- Nous verrons ultérieurement les passages de paramètres à la fonction main.

Premier exercice

- Reprendre le thème du premier exemple, c'est à dire écrire un programme qui affiche une chaîne de caractères à l'écran
- Le compiler et l'exécuter.

Déclarations

Préprocesseur directive #include

- Cette directive permet d'incorporer, avant compilation, le texte figurant dans un fichier quelconque. Elle peut être utilisée aussi bien pour pour des fichiers "en-têtes" prédéfinis que pour des fichiers "en-têtes" personnalisés.
- Cette directive possède deux syntaxes voisines :
  `#include <nom_fichier>`
- recherche le fichier mentionné dans un emplacement (chemin,répertoire) défini par l'implémentation (variable -I du compilateur).
  `#include "nom_fichier"`
- recherche le fichier mentionné dans le même emplacement (chemin, répertoire) que celui où se trouve le programme source.

Variables

- Type varnom1,varnom2;
- Une variable est une donnée qui peut avoir plusieurs valeurs
- int x=3 ; veut dire que x prend la valeur 3
- x=x+1; veut dire que x prend comme nouvelle valeur, sa valeur incrémenté d'une unité
- **Exemple :**

```
int i,j;
int u=3;
char c='\n';
```

- Attention à ne pas utiliser les mots clés comme nom de variable

Portée et initialisation des variables

- La variable n'est connue que dans le bloc où elle est déclarée (c'est à dire entre les accolades)
- les variables globales sont déclarées en dehors du bloc main
- On initialise une variable par une constante, une variable déjà initialisée, le retour d'une fonction

```
int i=1;
int j=i;
int k=carre(i);
```

- Syntaxe composite:

```
int i,j,k ;
i=j=k=1;
```

Portée des variables

- Sachant que le programme printf que nous n'avons pas encore vu permet d'afficher un entier `i` via la syntaxe :
  `printf ("La valeur de i est  %d\n",i)`
- Que l'on déclare une valeur entière qui vaut la valeur 2 par la syntaxe
  `int i=2;`
- Déclarer dans votre programme main.c plusieurs blocs imbriqués, dans chacun des blocs déclarer une variable i avec une valeur différente, l'afficher.
- Quelles sont les conclusions ?

Types

Rubriques

- Types de base
- Constance

Types de base

Type caractères : `char`

- Le type `char` est utilisé pour représenter un caractère, plus précisément la valeur entière d'un élément de l'ensemble des caractères représentables.
- Ce nombre entier est le code ASCII initialement sur 7 bits (American Standard for Information Interchange) du caractère. Un caractère unique occupe un octet. A est le caractère 65, a est 97.
- Le jeu de caractères ISO-8859 (sur 8 bits) :
  - les 128 premiers caractères correspondent aux caractères ASCII.
  - Les 128 derniers caractères (codés sur 8 bits) sont utilisés pour les caractères propres aux différentes langues.
  - La version ISO-8859-1 (aussi appelée ISO-LATIN-1) est utilisée pour les langues d'Europe occidentale.
  - Ainsi, le caractère de code 232 est le è.

Type entier : `int`

- Le type `int` a une taille de représentation qui dépend du système sur lequel on travaille.
- Par exemple, sur un vieux pc 8086 un élément de type `int` est stocké sur 2 octets (16 bits)
  alors que sur un processeur I5, il occupe 4 octets (32 bits)
- Sur une architecture 64 bits on garde 32 bits pour les entiers pour des raisons de compatibilité du code
- Le type `short` permet de stocker des données sur 2 octets.
- Le type `long` permet de stocker des données sur 32 bits si le type entier est sur 16 ou sur 64 si entier est sur 32.
- Si on ajoute le préfixe `unsigned` à la définition d'un type de variable entière, le bit de poids fort ne sert plus au signe mais augmente la portée d'une unité (ex : unsigned short : 0 -> 65532)

Les flottants : `float`

- Un nombre du type `float` est habituellement rangé en mémoire sous la forme suivante : mantisse sur 23 bits, exposant sur 8 bits et signe sur 1 bit .3,4 * 10-38 à 3,4 * 10 38
- Le type `double` est stocké sur 8 octets. Il est codé comme le type float, à ceci près que sa mantisse est sur 52 bits et son exposant sur 11 bits. 1,7*10-308 à 3,4* 10 308
- `Long double` mantisse de 64 bits et un exposant de 15 bits 3,4 * 10 -4932 à 3,4 * 10 4932
- Ici encore la taille dépend du processeur comme pour les entiers.

Déclaration des variables

- Pour chaque type de base on peut définir une variable sous la forme (avec ou sans initialisation): `type nom`
- depuis 1999 on peut déclarer une variable locale un bloc non nécessairement en début de bloc mais après des instructions.

```c
/*type nom ; */
int i;
float val ;
char c ='c';
long double k ;
unsigned char d;
```

Constance

Constantes

- Entière 12 , -3 ,06 (caractère zéro octal), 0xFF (hexadécimal), 12L (type long)
- Flottant : 2.897,15.1E-2, .0089e2,-4005 E3
- Caractères : 'a', 'Z', '\"
- \n : nouvelle ligne \t : tabulation \b : retour arrière
- \r retour chariot \f saut de page \a signal sonore
- \" :guillemet
- \\ : \
- \ddd code ASCII en notation décimale
- \xddd code ASCII en hexadécimal
- Chaînes de caractères : "\nCeci est une chaine"

Rappel binaire octal décimal hexadécimal

Binaire	Octal	Héxadécimal	Décimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1111	17	F	15

Déclaration de variable constante

- une variable d'un des types numériques peut être déclarée constante par adjonction du mot `const` devant son type :
- **Exemple :**

```
const int i =3;
```

- Une fois cette variable constante déclarée elle ne peut pas changer de valeur.
- Une variable constante ne peut être affectée à une valeur donnée qu'à sa déclaration.

Opérateurs

Rubriques

- Opérateurs simples
- Autres opérateurs
- Opérateurs avancés

Opérateurs simples

Opérateurs d'affectation =

- variable = expression ;
- le terme de gauche prend la valeur à droite
  - le terme de gauche ne peut pas être une constante
  - le terme de droite peut être une variable ou une constante

```
i = j + 2;
```

- La notation `int i=2, j = 2;` est équivalente à `int i=2; int j=2;`
- La notation `i=j=k=valeur ;` est équivalente à `i=valeur; j=valeur; k=valeur;`
- **Exemple :** int a=3;

Opérateurs arithmétiques

- Les opérateurs arithmétiques classiques comprennent :
  - l'opérateur unaire - (changement de signe)
  - les opérateurs binaires
    - + addition
    - - soustraction
    - * multiplication
    - / division (entière entre entiers et flottante entre flottants)
    - % reste de la division (modulo)
- 3/2 => 1 alors que 3/2. => 1.5
- **Exemple :** a=b+3;

NB : pas d'opérateur exposant mais la fonction pow(x,y)

Les conversions numériques implicites

- On peut ajouter un entier à un char :

```
char car='A' ;
car=car+1 ;
/* car vaut 'B' */
```

- Conversion implicite d'un type flottant vers un entier :

```
float val=2.97;
int i = val;
/* i vaut 2 */
```

Exercice opérateur

- Peut-on faire ceci ?

int i = 3.7;

- Que se passe-t-il si on fait cela juste après ?

int i= 3.7+7.3 ;

- Que donne ?

i=3.7+7.3;

Les opérateurs relationnels

- $>$ strictement supérieur
- $>=$ supérieur ou égal
- $<$ strictement inférieur
- $<=$ inférieur ou égal
- $==$ égal
- $!=$ différent
- Les retours de ces expressions sont des entiers (1 pour vrai , 0 pour faux) car il n'y a pas de booléens en C
- **Exemple :** if (a == 3)

Autres opérateurs

Les opérateurs logiques

- Ces opérateurs sont :
  - && et logique
  - || ou logique
  - ! négation logique
- Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un int qui vaut 1 si la condition est vraie et 0 sinon.
- L'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est évalué.
- **Exemple :**

```
if ( a==2 || a==3 )
```

Les opérateurs de manipulation de bits

- Les six opérateurs suivants permettent de manipuler des entiers au niveau du bit.
  - & et : 1100 & 1001 => 1000
  - | ou inclusif : 1000 | 1001 => 1001
  - ^ ou exclusif : 1000 ^ 1001 => 0001
  - ~ complément à 1 : ~ 1000 => 0111
  - << décalage à gauche : 1001 << 1 => 0010
  - >> décalage à droite : 1001 >> 1 => 0100
- Ils s'appliquent aux entiers de toutes longueurs (short, int ou long), signés ou non.

Les opérateurs d'incrémentation et de décrémentation

- Les opérateurs d'incrémentation ++ et de décrémentation – s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i).
- Dans les deux cas la variable i sera incrémentée :
  - dans la notation suffixe la valeur retournée sera l'ancienne valeur de i
  - dans la notation préfixe se sera la nouvelle.
- **Exemple :**

```
int a = 3, b, c;
b = ++a;        /* a et b valent 4 */
c = b++;        /* c vaut 4 et b vaut 5 */
```

Affectation composée

- Les opérateurs d'affectation composée sont `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`
- Pour tout opérateur op, l'expression expression1 op= expression2 est équivalente à expression1 = expression1 op expression2
- **Exemple :**

```
a+=3 ; /* est équivalent à a=a+3; */
```

Opérateur virgule

- Une expression peut être constituée d'une suite d'expressions séparées par des virgules : expression1, expression2, . . . , expressionn
- Cette expression est alors évaluée de gauche à droite. Sa valeur sera la valeur de l'expression de droite.
- **Exemple :**

```
main()
{
  int a, b;
  b = ((a = 3), (a + 2));
  printf("\n b = %d \n",b);
}
/*imprime b = 5.*/
```

Opérateurs avancés

Opérateur conditionnel ternaire

- Sa syntaxe est la suivante : condition ? expression1 : expression2
- Cette expression est égale à expression1 si condition est satisfaite, et à expression2 sinon.
- **Exemple :**

```
x=x >= 0 ? x : -x
```

est équivalent à

```
if ( x >= 0) x=x; else x=-x;
/* C'est la valeur absolue */
```

Les opérateurs de cast

- L'opérateur de conversion de type, appelé cast, permet de modifier explicitement le type d'un objet.
- On écrit `(type) objet`
- **Exemple :**

```
main()
{
  int i = 3, j = 2;
  printf("%f \n",(float)i/j);
}
/* retourne 1.5*/
```

L'opérateur &

- L'opérateur d'adresse & appliqué à une variable retourne l'adresse-mémoire de cette variable.
- La syntaxe est &objet
- **Exemple** :

```
int *p=&variable
```

- Ne s'applique pas à une constante.

L'opérateur sizeof

- retourne la taille mémoire occupée par l'objet passé en argument
- La syntaxe est sizeof(x) ou sizeof (type)
- **Exemple :**

```
long int i=18 ;
int s=sizeof(i);
# s vaut 4 ou  8 en fonction de l'architecture
```

Taille mémoire

- Afficher la taille mémoire prise par un entier, par une adresse sur un entier, par un long unsigned.

Priorité des opérateurs (ordre décroissant)

() [] -> .
! ~ ++ – - (type) * & sizeof * / %
+ -(binaire)
<< >>
< <= > >=
== !=
&(et bit-à-bit)
^
|
&&
||
? :
= += -= *= /= %= &= ^= |= <<= >>=
,

Contrôles

Rubriques

- Instructions conditionnelles
- Boucles

Instructions conditionnelles

if—else

- La forme la plus générale comprend un nombre quelconque de `else if ( ... )`
- Le dernier `else` est toujours facultatif.

```
if (expression1 )
  instruction1
else if (expression2 )
  instruction2
else if (expressionN )
  instructionN
else
  instruction
```

switch

```
switch (expression )
 {case constante1:
   liste d'instructions 1 ; break;
 case constanteN:
   liste d'instructions N ; break;
 default:
   liste d'instructions ;  break;
 }
```

- Si la valeur de expression est égale à l'une des constantes, la liste d'instructions correspondant est exécutée.
- Sinon la liste d'instructions correspondant à `default` est exécutée. L'instruction `default` est facultative.

Remarque : si les parties entre case contiennent des déclarations de variables, il faudra encadrer la liste d'instruction par des {} comme suit :

Branchement non conditionnel break

- L'instruction break est employée à l'intérieur de n'importe quelle boucle.
- Elle permet d'interrompre le déroulement de la boucle et passe à la première instruction qui suit la boucle.
- En cas de boucles imbriquées, break fait sortir de la boucle la plus interne. Par exemple, le programme suivant :

```
int i;
for (i = 0; i < 5; i++)  {
    printf("i = %d\n",i);
    if (i == 3)
      break;
  }
printf("valeur de i a la sortie de la boucle = %d\n",i);
```

affiche à l'écran les valeurs de 0 à 3

Branchement non conditionnel continue

- L'instruction continue permet de passer directement au tour de boucle suivant, sans exécuter les autres instructions de la boucle.

```
int i;
for (i = 0; i < 5; i++)
  {
    if (i == 3)
      continue;
    printf("i = %d\n",i);
  }
printf("valeur de i a la sortie de la boucle = %d\n",i);
```

affiche à l'écran 0 1 2 4

Branchements non conditionnel goto

- L'instruction `goto` permet d'effectuer un saut jusqu'à l'instruction `etiquette` correspondant.
- Elle est généralement peu recommandée mais on la trouve dans des cas d'imbrications multiples.
- Une étiquette est définie par un identifiant suivi de :
- **Exemple :**

```
    while (a) {
        while (b) {
            if (c)
                goto end ;
            ...
        }
end:
    ...
```

Boucles

Boucle while

- La syntaxe de while est la suivante :

```
while (expression )
  instruction
```

- Tant que expression est vérifiée (i.e., non nulle), instruction est exécutée.
- Si expression est nulle au départ, instruction ne sera jamais exécutée.
- instruction peut évidemment être une instruction composée.
- **Exemple :**

```
i = 1;
while (i < 10)  {
    printf("\n i = %d",i);
    i++; }
```

Boucle do—while

- Il peut arriver que l'on ne veuille effectuer le test de continuation qu'après avoir exécuté l'instruction.
- Dans ce cas, on utilise la boucle `do---while`. Sa syntaxe est

```
do
  instruction
while (expression );
```

- Ici, instruction sera exécutée tant que expression est non nulle.
- instruction est exécutée au moins une fois.
- **Exemple :**

```
int a;
do  {    printf("\n Valeur entière : %d\n",a++);  }
while ((a <= 0) || (a > 10));
```

Boucle for

- La syntaxe de for est :

```
for (expr 1 ;expr 2 ;expr 3)
  instruction
```

- **Exemple :**

```
int i;
for (i = 0; i < 10; i++)
  printf("\n i = %d",i);
for (int j = 0; j < 10; j++)
  printf("\n j = %d",j);
```

- À la fin de cette boucle, i vaudra 10.
- Les trois expressions utilisées dans une boucle for peuvent être constituées de plusieurs expressions séparées par des virgules.

Codes ASCII

- Afficher le code ASCII des caractères de a à z
- Afficher ensuite les caractères ASCII A à z

IO

Rubriques

- Sortie
- Entrées
- Tableaux
- Chaînes de caractères
- Fonctions sur chaînes de caractères

Sortie

printf

- La fonction est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi.
- Sa syntaxe est

```
printf ("chaîne de contrôle ",expr1, ..., exprN);
```

- La chaîne de contrôle contient le texte à afficher et les spécifications de format correspondant à chaque expression de la liste.
- Les spécifications de format ont pour but d'annoncer le format des données à visualiser.
- Elles sont introduites par le caractère %, suivi d'un caractère désignant le format d'impression.

Format d'impression

- %d int décimale signée (%ld pour long)
- %u unsigned int décimale non signée (%lu pour long)
- %o unsigned int octale non signée (%lo pour long)
- %x unsigned int hexadécimale non signée (%lx pour long)
- %f double décimale virgule fixe (%lf pour long)
- %e double décimale notation exponentielle (%le pour long)
- %g double décimale, représentation la plus courte parmi %f et %e (%lg pour long)
- %c unsigned char caractère
- %s char* chaîne de caractères

Précision des caractères d'impression

- En plus du caractère donnant le type des données, on peut éventuellement préciser certains paramètres du format d'impression, qui sont spécifiés entre le % et le caractère de conversion dans l'ordre suivant :
  - Largeur minimale du champ d'impression : %10d spécifie qu'au moins 10 caractères seront réservés pour imprimer l'entier.
  - Par défaut, la donnée sera cadrée à droite du champ.
  - Le signe - avant le format signifie que la donnée sera cadrée à gauche du champ (%-10d).
- précision : %.12f signifie qu'un flottant sera imprimé avec 12 chiffres après la virgule.
- De même %10.2f signifie que l'on réserve 12 caractères (incluant le caractère .) pour imprimer le flottant et que 2 d'entre eux sont destinés aux chiffres après la virgule.

Précision (suite)

- Lorsque la précision n'est pas spécifiée, elle correspond par défaut à 6 chiffres après la virgule.
- Pour une chaîne de caractères, la précision correspond au nombre de caractères imprimés : %30.4s signifie que l'on réserve un champ de 30 caractères pour imprimer la chaîne mais que seulement les 4 premiers caractères seront imprimés (suivis de 26 blancs).

Entrées

Exemples

```
int i = 23674;  int j = -23674;
long int k = (1l << 32);
double x = 1e-8 + 1000;
char c = 'A';
char *chaine = "chaine de caracteres";
printf("impression de i: \n");
printf("%d \t %u \t %o \t %x",i,i,i,i);
printf("\nimpression de j: \n");
printf("%d \t %u \t %o \t %x",j,j,j,j);
printf("\n%.2f \t %.2e",x,x);
printf("\n%.20f \t %.20e",x,x);
printf("\nimpression de c: \n");
printf("%c \t %d",c,c);
printf("\nimpression de chaine: %s\t %.10s \n",chaine,chaine);
```

Caractères ASCII

- Afficher les caractères ASCII de A à z en affichant cette fois leur valeur ASCII et leur valeur caractère.

scanf

- La fonction permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonctions.

```
scanf ("chaîne de contrôle",argument1,...,argumentN)
```

- La chaîne de contrôle indique le format dans lequel les données lues sont converties.
- Elle ne contient pas d'autres caractères (pas de \n).
- Comme pour printf, les conversions de format sont spécifiées par un caractère précédé du signe %.
- On peut fixer le nombre de caractères de la donnée à lire : % 3s pour une chaîne de 3 caractères, % 3d pour un entier de 10 chiffres

Exemple scanf

```
#include <stdio.h>
main()
{
  int i;
  printf("entrez un entier sous forme hexadecimale i = ");
  scanf("%x",&i);
  printf("i = %d\n",i);
}
```

- Si on entre au clavier la valeur 1a, le programme affiche i = 26.

Putchar et getchar

- Les fonctions `getchar` et `putchar` permettent respectivement de lire et d'imprimer des caractères.
- Il s'agit de fonctions d'entrées-sorties non formatées.
- La fonction `getchar` retourne un `int` correspondant au caractère lu. Pour mettre le caractère lu dans une variable `caractere`, on écrit `caractere = getchar();`
- Lorsqu'elle détecte la fin de fichier, elle retourne l'entier EOF (End Of File), valeur définie dans la librairie stdio.h.
- La fonction `putchar` écrit `caractere` sur la sortie standard : `putchar(caractere);`

Nombre d'or

- Définir un entier entrer 0 et 100 appelé nombre d'or
- Demander à un joueur d'entrer un nombre et à chaque réponse

indiquer si le nombre d'or est plus petit ou plus grand

- Une fois que le nombre a été trouvé afficher en combien de passes

le jeu a été gagné

- Ceux qui auront du temps pourront se documenter sur la fonction rand

pour produire le nombre d'or de façon aléatoire.

Tableaux

Tableaux

Présentation

- Dans certaines applications, il serait utile de pouvoir ranger des valeurs logiquement rattachées, non pas dans des variables distinctes mais dans une variable commune.
- Par exemple, si on veut stocker en mémoire les notes obtenues par 50 étudiants à un examen, il serait plus simple de pouvoir stocker toutes ces notes dans une variable de type complexe plutôt que de devoir les stocker dans 50 variables différentes.
- Les variables qui permettent de stocker non pas une mais plusieurs valeurs sont des variables de types complexes(composés).
- Le langage C dispose de deux types de données complexes : les tableaux et les structures.
- Le tableau est unidimensionnel lorsque ses éléments ne sont pas eux-mêmes des tableaux.

Tableaux à une dimension

- La définition d'un tableau unidimensionnel s'effectue via la syntaxe suivante :

```
<Type> <Nom du tableau> [nombre d'éléments];
```

- **Exemple :**

```
int t[4];
```

- contient donc 4 éléments : t[0], t[1], t[2] et t[3]
- Utilisation : `t[2]=7;`
- Les compilateurs C ne vérifient pas si l'index est valide, donc compris entre les bornes 0 et index maximal.
- Si on emploie des indices qui dépassent le maximum autorisé, alors on adresse des emplacements de mémoire situés hors du tableau sans que le compilateur ne signale d'erreurs

Initialisation des tableaux uni-dimensionnels

-
- Méthode 1 :

```
int x[3];
x[0]=x[1]=x[2]=0;
```

- Méthode 2 : Au cours de sa définition

```
<Type> <Nom du tableau> <[nombre d'éléments>]={k1,k2,...,kn};
```

- **Exemple :** `int t[5]={0,0,0,0,0};`
- qui peut aussi s'écrire `int t[5]={0}` ;car tous les éléments sont identiques

Tableaux

- Afficher un tableau qui comprend 1 au premier rang et le double au rang suivant. 1, 2, 4, 8 etc..
- On limitera l'exercice au 30 premiers entiers.

Que se passe-t-il si on va jusqu'aux 40 premiers entiers ?

Tableaux multi-dimensionnels

```
<Type> <Nom du tableau> [e1] [e2] ... [en];
```

- **Exemple :** `int k[3][4];`
- Pour un tableau à deux dimensions, le premier indice est l'indice de ligne et le second indice est l'indice de colonne
- Contrairement à ce qu'il se passe avec les tableaux unidimensionnels, pour les tableaux multidimensionnels l'image par laquelle on les représente habituellement ne concorde plus avec leur rangement en mémoire. Un tableau bidimensionnel est généralement représenté sous la forme d'une matrice rectangulaire. En mémoire, par contre, un tel tableau n'est nullement rangé sous forme de "rectangle".
- Au contraire, les éléments des tableaux même multidimensionnels sont toujours disposés les uns à la suite des autres.

Initialisation des tableaux multi-dimensionnels

- On peut procéder comme pour les tableaux unidimensionnels en indiquant les valeurs d'initialisation à droite de l'opérateur d'affectation "=", entre des accolades et séparées les unes des autres par des virgules :

```
int k[2][3]={700,500,200,900,800,400};
```

- Les valeurs sont attribuées aux éléments du tableau dans l'ordre. Le premier élément k[0][0] reçoit la première valeur de la liste (700),
- Le second élément k[0][1] reçoit la seconde valeur de la liste (500) ...
- On préfère l'initialisation sous la forme:

```
int k[2][3]={ {700,500,200},{900,800,400}};
```

Chaînes de caractères

Les chaînes de caractères

- (ou string) sont des suites de caractères composées de signes faisant partie du jeu de caractères représentables de l'ordinateur.
- qu'elles soient constantes ou variables, elles sont de par leur structure des tableaux à une dimension ayant des éléments de type char.
- La définition suivante crée un tableau s de 20 éléments de type char.

```
char s[20];
```

- Cette chaîne 's' pourra en fait contenir au maximum 19 caractères et pas 20 car le dernier caractère est 0

Initialisation d'une chaîne de caractères

```
char chaine1[41];
scanf("%s",chaine1); /* et pas &chaine*/
char *chaine2="comment allez vous?";
char chaine3[]="bonjour";
char chaine4[]={'b','o','n','j','o','u','r','\0'};
while((name[i]=getchar())!='\\n'){
            i++;
          chaine5[i]='\\0'; }
```

- Nous verrons que les chaînes de caractères sont également des pointeurs : s[] se note *s

Copie de chaînes de caractères

- Il existe une fonction de bibliothèque strcpy qui permet de recopier une chaîne dans un tableau char, ce qui correspond bien à la manipulation demandée.
  `strcpy(<tableau1destination>,<tableau2source>);`
- La fonction strcpy copie le contenu du tableau dont l'adresse est spécifiée par le second paramètre de la fonction (caractère nul compris) dans le tableau dont l'adresse est donnée par le second paramètre.
- Le tableau "cible" doit prévoir la place mémoire :

```
char s[8];
strcpy(s,"bonjour");
```

- L'utilisation de strcpy et de quelques autres fonctions de traitement de chaînes exige un `#include <string.h>`

Concaténation de chaînes de caractères

```
char *strcat (char *chaine1 , char * chaine2 );
```

- Retourne un pointeur sur chaine1
- Chaine1 est concaténé avec chaine2
- Ne pas oublier de réserver assez de mémoire pour chaine1
- **Exemple :**

```
char ch[100];
strcpy(ch,"bonjour");
strcat(ch," gilles");
```

Fonctions sur chaînes de caractères

Comparaison de chaînes de caractères

```
int strcmp(char * ch1, char *ch2 );
```

- si la valeur est inférieure à 0, alors la chaîne en premier paramètre de la fonction est lexicographiquement inférieure à l'autre chaîne
- si la valeur est égale à 0, les chaînes sont égales
- si la valeur est supérieure à 0, alors la chaîne 2 est lexicographiquement supérieure à l'autre chaîne.

Autres opérations sur les chaînes de caractères

- longueur d'une chaîne de caractères :

```
int =strlen( char * str);
```

- copie n caractères

```
char *strncpy(char *dest, const char *src, size_t n);
```

Recherche dans une chaîne

- **strchr :** `char *strchr(char *chaine, char c)` : retourne un pointeur sur la première occurence de c dans chaine, et NULL si c n'y figure pas.
- **strrchr :** `char *strrchr(char *chaine, char c)` : retourne un pointeur sur la dernière occurence de c dans chaine, et NULL si c n'y figure pas.
- **strstr :** `char *strstr(char *ch1, char *ch2)` : retourne un pointeur sur la première occurence de ch2 dans ch1, et NULL si ch2 n'y figure pas.

Fonctions

Rubriques

- Paramètres

Introduction

Les fonctions

- Dans un programme, on est souvent amené à répéter plusieurs fois les mêmes opérations à des endroits différents et avec des données différentes.
- Pour éviter de devoir réécrire chaque fois cette suite d'opérations, la plupart des langages de programmation permettent au programmeur de définir des sous-programmes qu'il suffit d'appeler aux endroits voulus.
- Ces sous-programmes sont aussi très utiles lorsqu'on a un problème très complexe à traiter. En effet, il est plus facile de le découper en sous-problèmes et d'écrire un sous-programme pour chacun d'eux.
- Il suffit alors de rassembler le tout dans un programme principal. En C tous les sous-programmes sont appelés des fonctions.

Définition d'une fonction

- Pour construire une fonction, il faut coder les instructions qu'elle doit exécuter :

```
type identificateur (type par1, type par2)
{
    corps de la fonction
}
```

- S'il n'y a pas de paramètres, les parenthèses sont quand même obligatoires
- Les définitions de fonction se placent dans n'importe quel ordre mais toujours en dehors de toute fonction
- Si la fonction n'est pas définie avant son appel, on est obligé de la déclarer.

Exemple de déclaration et d'utilisation d'une fonction

- Définition et appel dans un même fichier

```
#include <stdio.h>

void message() {
  printf("cours de langage C\n");
}

void main()   {
   message();
}
```

- Ici la définition de la fonction est faite avant l'appel

Autre exemple

- Sur certains compilateurs la séquence suivante pose problème

```
#include <stdio.h>

void main()   {
   message();
}
void message() {
  printf("cours de langage\\n");
}
```

- on placera la déclaration avant le main `void message(); /*avec les arguments éventuels */`

Déclaration de fonction

- En général la fonction est définie indépendamment du code appelant
- On déclare en effet une fonction définie ailleurs avec ses paramètres par `type identificateur (liste des types des paramètres);`
- Ceci se fait généralement au sein d'un fichier d'extension h dans un fichier include que l'on insère par le code `#include fichiercontenantpleindefonction.h"`

Paramètres

Paramètres de fonction

- **Exemple :**

```
 void  traceligne(int x0, int y0, int x1, int y1)
{
    /* séquence de programme affichant une ligne
    les variables x0, y0, x1, y1 sont utilisées
    comme des variables locales
    */
}
```

- Un paramètre formel est une variable locale, connue seulement à l'intérieur de la fonction où est défini ce paramètre.
- Les paramètres formels et effectifs doivent correspondre en nombre et en type à ceux fournis à l'appel, les noms par contre peuvent différer.

Retour d'une fonction

- L'instruction `return` met fin à l'exécution des instructions d'une fonction et rend le contrôle du programme à l'instance appelante.
- Éventuellement la fonction peut renvoyer une valeur que l'instance concernée peut utiliser dans ses propres instructions.
- `return (<expression>)`
- Les parenthèses sont facultatives.
- Dans ce cas, la fonction ne doit plus avoir un type void mais un type en corrélation avec le type retourné :

```
int cube(int x)
{
    return ( x*x*x);
}
```

Appel d'une fonction

- L'appel se fait en donnant le nom suivi des arguments entre parenthèse.
- Exemple avec la fonction cube définie précédemment :

```
int valeur=cube(23);
```

- Les valeurs des paramètres effectifs sont recopiées dans les paramètres formels correspondants.
- Si ces valeurs sont modifiées dans la fonction, elles ne le seront pas dans la fonction appelante (passage par valeur).

Exemple didactique

- Soit la fonction

```
int mafunction( int x )
{
  x=(x-1)*(x+1);
  return x ;
}
```

- Étudier l'appel suivant :

```
int x=3;
int y =mafunction(x);
/* ici y vaudra 8, mais x vaudra 3 */
```

Exercices fonctions

- Écrire la fonction max qui renvoie le maximum de deux nombres,
- Écrire la fonction min qui renvoie le minimum de deux nombres
- Écrire la fonction ascii qui renvoie le code ASCII d'un caractère
- Écrire la fonction Char qui renvoie le caractère correspondant à

un entier entre 0 et 255

- Écrire la fonction premmajuscule qui met le premier caractère d'une

chaîne de caractères en majuscule.

- Écrire la fonction majuscule qui met une chaîne de caractères en

majuscule,

- Écrire la fonction minuscule qui met une chaîne de caractères en

minuscule

- Les tester dans le corps de la fonction main()

Passage de l'adresse à une fonction

- En C, une fonction ne peut retourner qu'une seule valeur.
- De base, une fonction ne peut pas modifier la valeur de ses paramètres
- Nous reviendrons quand nous étudierons les pointeurs sur une façon de permettre ces modifications d'argument

Directives

Rubriques

- Directives simples
- Compilation conditionnelle
- Autres

Directives simples

Les directives de compilation

- #include : inclut le fichier à l'emplacement de la directive
- #define : permet de définir une constante de préprocesseur.
- #define macro() : permet de définir la macro
- #undef : supprime la définition actuelle
- #ifdef #ifndef #endif : condition
- #if #elif #else #endif : condition
- variables définies
- #error

Les #define

```
#define nbmax 5
```

- demande de substituer au symbole nbmax le texte 5, et cela à chaque fois que ce symbole apparaîtra dans la suite du fichier source.

```
#define entier int
```

- placée en début de programme, la directive permet d'écrire "en français" les déclarations de variables entières.
- Ainsi `entier a,b;` sera remplacé par `int a,b;`
- On peut également remplacer une instruction par un mot :

```
#define bonjour printf("bonjour");
#define affiche printf("resultat %d\\n",a);
#define ligne printf("\\n");
```

Attention attention

- Une directive définit une constante pas une variable ainsi :

```
#define MINIMAL 10
int i=10;;
i=MINIMAL+10; /*i vaut 20*/
MINIMAL=MINIMAL+10; /* erreur du compilateur */
#define MINIMAL = 10 /* ne passe pas */
#define MINIMAL 10;/* très dangereux mais passe*/
```

Macro avec argument

- Une macro avec paramètres se définit de la manière suivante :

```
#define nom(liste-de-paramètres) corps-de-la-macro
```

où liste-de-paramètres est une liste d'identificateurs séparés par des virgules.

- **Exemple :**

```
#define MAX(a,b) (a > b ? a : b)
```

- Le processeur remplacera dans la suite du code toutes les occurrences du type `MAX(x,y)` où x et y sont des symboles quelconques par `(x > y ? x : y)`
- Une macro a donc une syntaxe similaire à celle d'une fonction, mais son emploi permet en général d'obtenir de meilleures performances en temps d'exécution.

Attention Attention

- Une macro n'est pas une fonction

```
#define MAX(x,y) x > y ? x : y
int valeur=2*MAX(2,3);
/*donne  2 *2 > 3 ? 2 : 3 => 4 > 3 ? 2 : 3 => 3*/
```

- si MAX était une fonction le résultat aurait été 6

Compilation conditionnelle

Directive #undef

- L'instruction `#undef` supprime la définition d'une constante symbolique précédemment créée via une instruction `#define`.
- Cela signifie qu'à partir de cet endroit-là la constante n'a plus aucun sens dans le programme.
- On peut de même supprimer la définition d'une macro :

```
#define ADD(a,b) a+b
/*...*/
#undef ADD
```

Compilation conditionnelle

- La compilation conditionnelle est la possibilité de compiler certaines partie du code suivant des conditions
- Ces conditions sont gérées par les directives
  - `#ifdef` (si un symbole est défini) `#ifndef` (s'il ne l'est pas)
  - ou par les directives `#if` `#elif` `#endif` plus précises permettant de tester des valeurs.
- Les deux premières conditions permettent de compiler une partie du code suivant l'existence de certaines conditions
- Les trois dernières permettent de compiler une partie du code en fonction de la valeur des conditions.

Directive #ifdef et #ifndef

- Les variables peuvent être définies de deux façons :
  - une option -D VALEUR ou -DVALEUR=valeur à été passée au compilateur
  - on a positionné des variables dans un fichier de configuration
- **Exemple :**
  - Dans un fichier de configuration

```
#define DEBUG
```

```
- Dans le corps du programme
```

```
#ifdef DEBUG
  for (i = 0; i < N; i++)
    printf("%d\n",i);
#endif
```

Directives #if #elif #endif

- La syntaxe générale la suivante (elif et else optionnels)

```
#if condition1
  partie-du-programme1
#elif condition2
  partie-du-programme2
     ...
#elif conditionN
  partie-du-programmeN
#else
  partie-du-programmeAutre
#endif
```

- Les conditions prendront des valeurs du préprocesseurs accompagnés des opérateurs logiques suivants `&&`, `||`, `!`, `==`, `<`, `<=`, `>`, `>=`

Autres

Symboles prédéfinis

Cette liste de symboles peut être renforcée par le modèle de votre compilateur

- __DATE__ : chaîne de caractère représentant la date de compilation.
- __TIME__ : chaîne de caractère représentant l'heure de compilation.
- __FILE__ : Nom du fichier source d'origine.
- __LINE__ : Ligne courante.
- __STDC_VERSION__ : permet de connaître la version du compilateur utilisé : entier long qui vaut 199901L si le compilateur respecte la norme C99.

Directive #defined

- Peut être utilisé sous l'une des deux formes suivantes : defined nom ou bien : defined (nom).
- Il délivre la valeur 1 si nom est une macro définie, et la valeur 0 sinon.
- L'intérêt de cet opérateur est de permettre d'écrire des tests portant sur la définition de plusieurs macros, alors que #ifdef ne peut en tester qu'une.

```
#if defined(SOLARIS) || defined(SYSV)
```

Directive #error

- La commande #error message : provoque erreur en affichant un message.
- La commande #warning affiche un message sans provoquer d'erreur
- La rencontre de cette commande provoquera l'émission d'un message d'erreur comprenant la suite-d-unités-lexicales.
- Cette commande a pour utilité de capturer à la compilation des conditions qui font que le programme ne peut pas s'exécuter sur cette plate-forme.

Message d'erreur

- Afficher un message d'erreur via #error si la variable TARGET n'est pas définie
- Si la variable TARGET est définie on l'affichera à l'écran