Algo & Info

Cours

Département Mesures Physiques

IUT Saint Nazaire

Organisation

- 6h CM [4 x 1,5h]
- 9h TD [6 x 1,5h]
- 28h SAé dont :
  - 5 séances C (x 3h)
  - puis 1 examen C (1h30)
  - puis 3 séances LabView (x 3h)
  - puis 1 examen LabView (1h30)
- Devoir surveillé en semaine 42 (Mi Octobre)
  - QCM, aucun document autorisé
- Autres TPs concernés :
  - PPP
  - Informatique d'instrumentation
  - Systèmes embarqués
  - Chaîne de mesures
  - Projets tutorés, ...

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d’un programme C
4. Ingrédients d’un algorithme

Plan

1. **Introduction**
   - Architecture minimale d'un ordinateur ou un microcontrôleur
   - Programme/Langage machine/Langage de programmation /Compilateur/IDE
   - Outils en ligne pour pratiquer le C

2. Algorithme et langage C

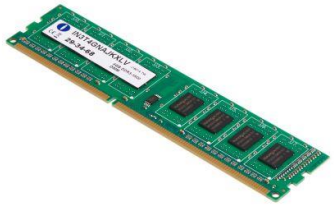
3. Structure d'un programme C

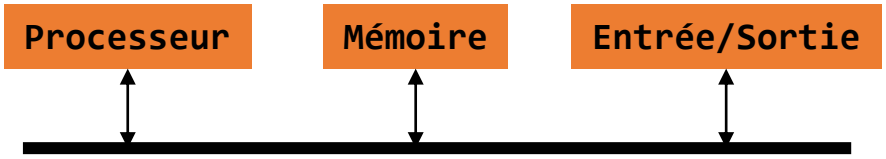
4. Ingrédients d'un algorithme

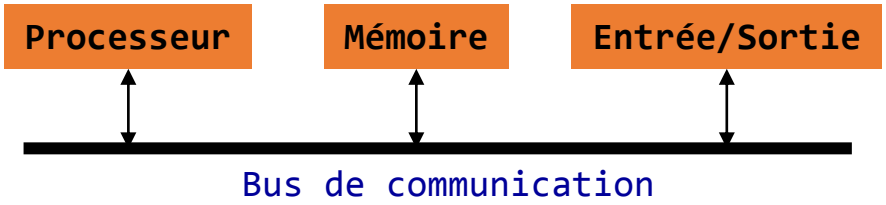
Ordinateur / Microcontrôleur

Architecture

Schéma simplifié





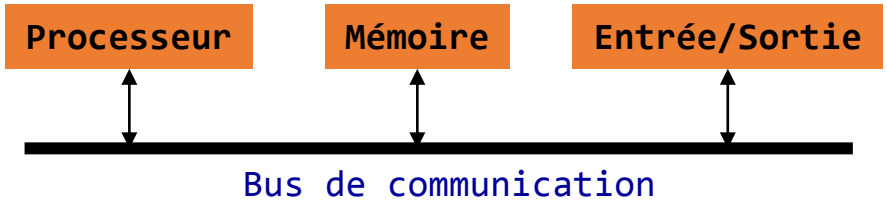


Processeur = unité de traitement = CPU (Central Processing Unit)

- **faire des calculs**
  - Instructions arithmétiques + - * /
  - Instructions de comparaison > < ==

- **manipuler l'information et les données provenant des entrées-sorties ou récupérée dans la mémoire.**
  - Les instructions d'accès mémoire.
  - Les instructions d'entrée-sortie avec les périphériques (clavier, écran, capteur, CAN, …).
  - …

Mémoire



Mémoire

- **2 catégories :**
  - Mémoire programme : stocker les programmes
  - Mémoire travail : stocker les valeurs des variables

un registre d'un octet = 8 bits

Chaque registre a une adresse.
Cette adresse permet de le sélectionner et de l'identifier celle-ci parmi tout les autres.

0	1	0	1	0	0	0	0	1
1	1	1	1	1	1	0	0	0
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	1	1
4	1	1	1	1	1	0	0	0
5	1	0	1	0	0	0	0	1

programme = une suite d'instructions dans l'ordre bien déterminé

données

- Le programme et les nombres seront tous codés en binaire dans la mémoire. C'est le seul langage que le processeur comprenne.

Ordinateur / Microcontrôleur

Compilateur

Programme en langage de programmation

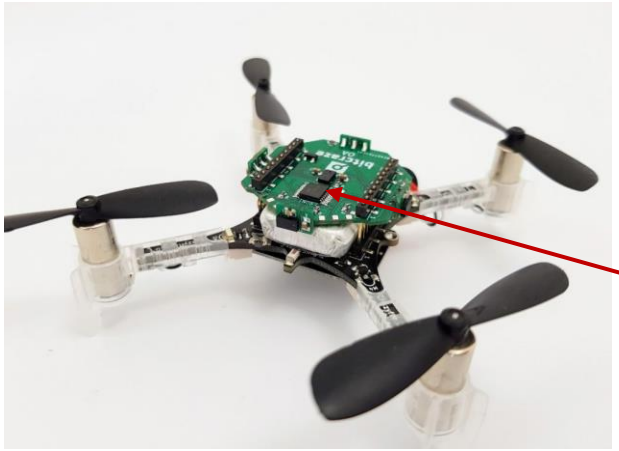
```
#include <stdio.h>

int main() {
    printf("Hello world \n");
    return 0;
}
```

Compilateur

Programme en langage machine

1	0	1	0	0	0	0	1
1	1	1	1	1	0	0	0
1	0	1	0	0	0	0	1
0	0	0	1	1	1	1	1

Bootloader



IDE

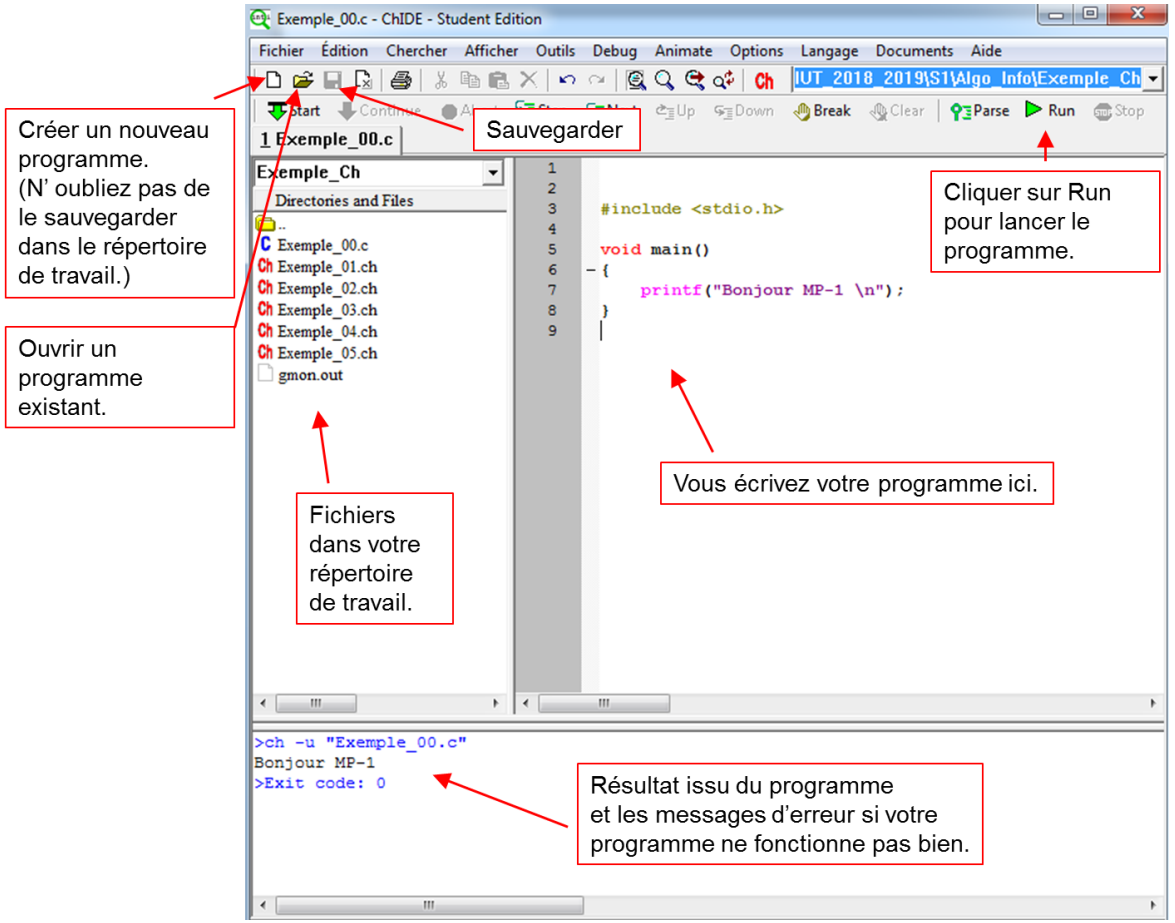
Integrated Development Environment

Environnement de développement intégré

IDE = Éditeur de texte + Compilateur + Débuggeur

Un IDE contient les outils permettant de :

- écrire un programme en langage de programmation
- Traduire ce programme en langage machine
- Exécuter ce programme
- Trouver les erreurs



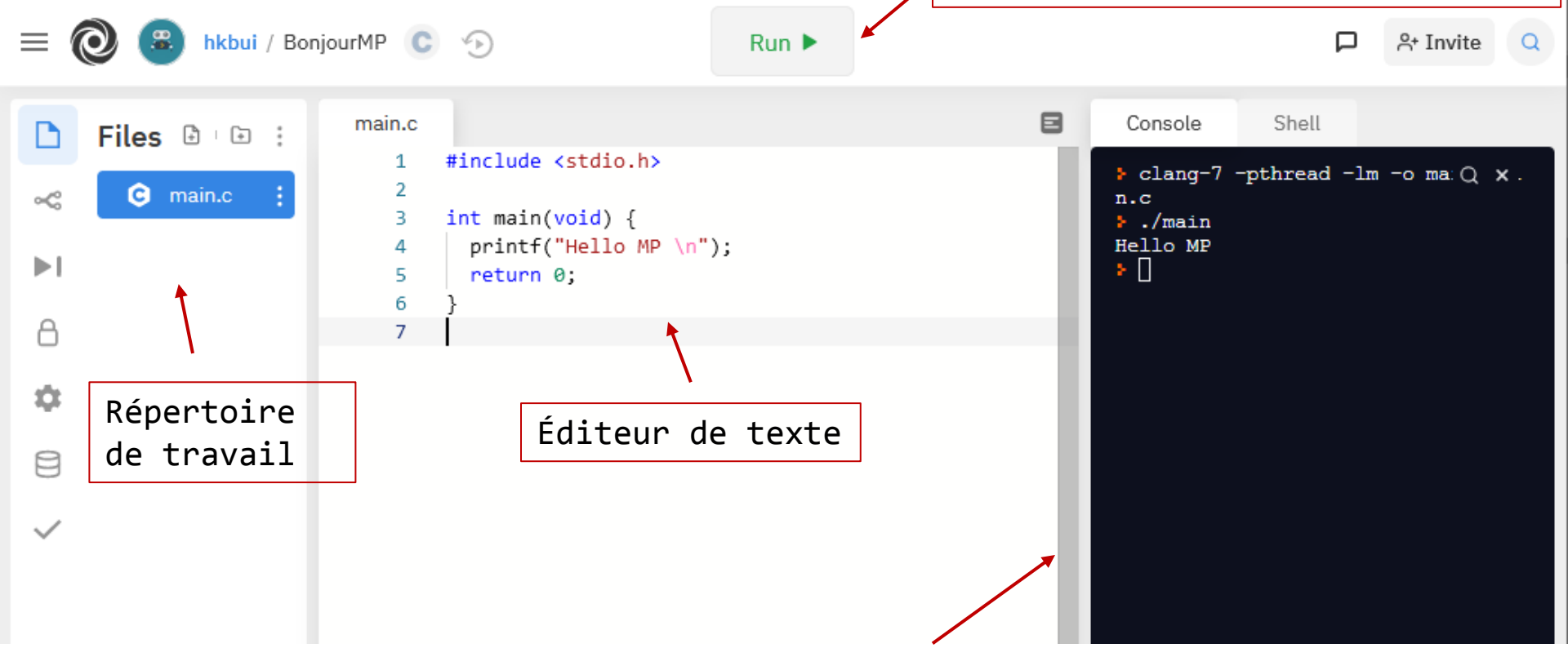
IDE + Compilateur en ligne

Compilateur en ligne pour pratiquer le C

replit.com

https://replit.com

Run = compilation + exécution



Répertoire de travail

Éditeur de texte

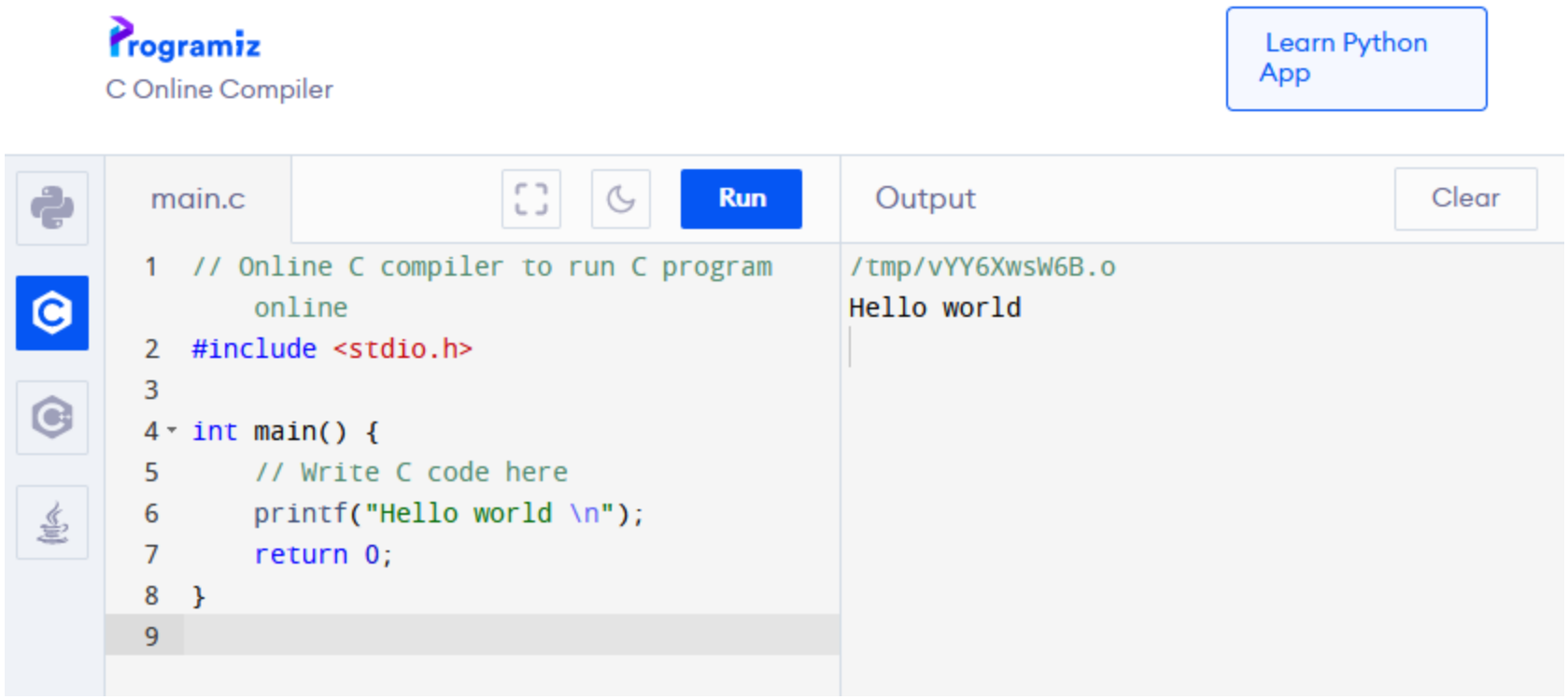
Console de terminal
où on voit les résultats affichés du programme

IDE + Compilateur en ligne

Compilateur en ligne pour pratiquer le C

programiz.com

https://www.programiz.com/c-programming/online-compiler/

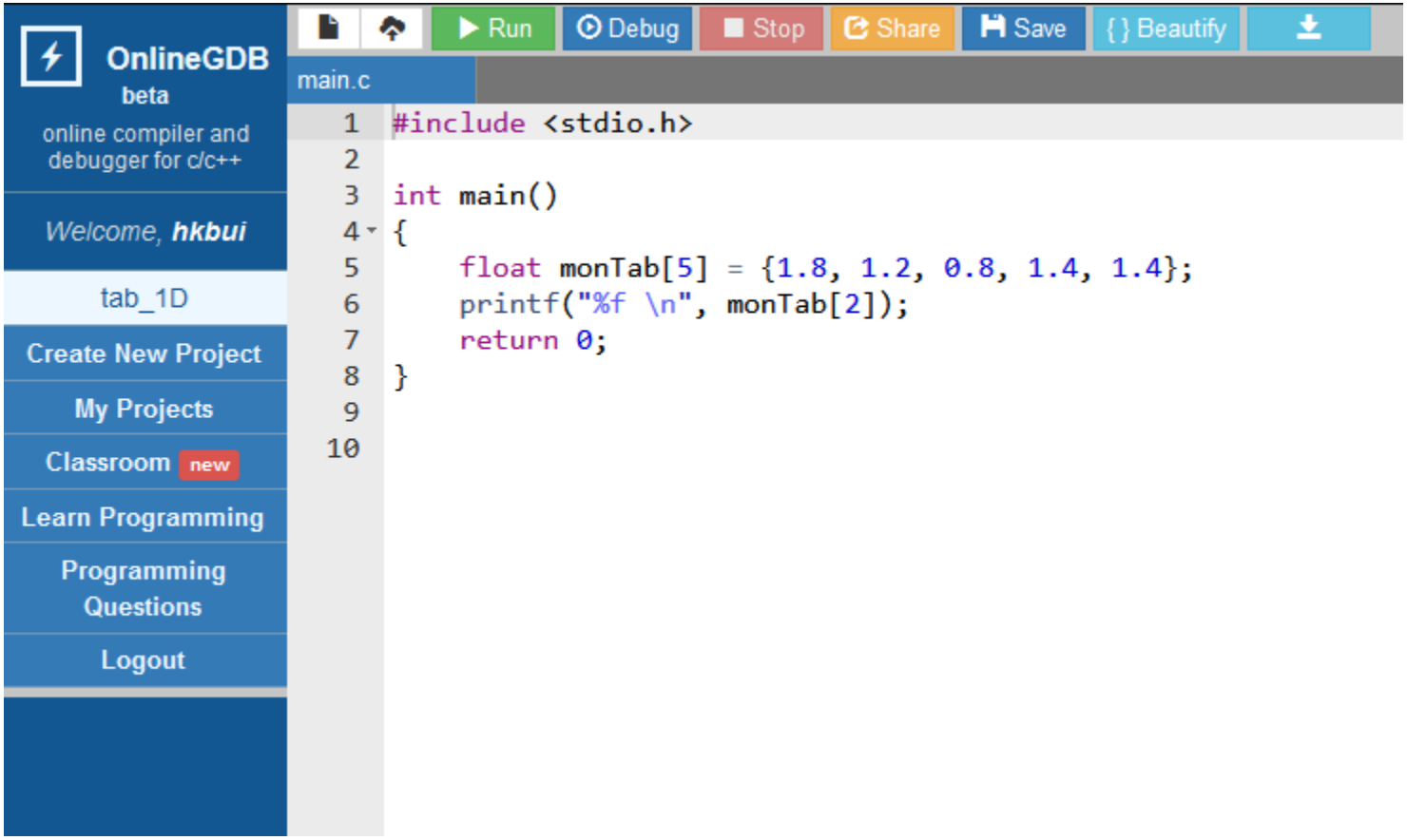


IDE + Compilateur en ligne

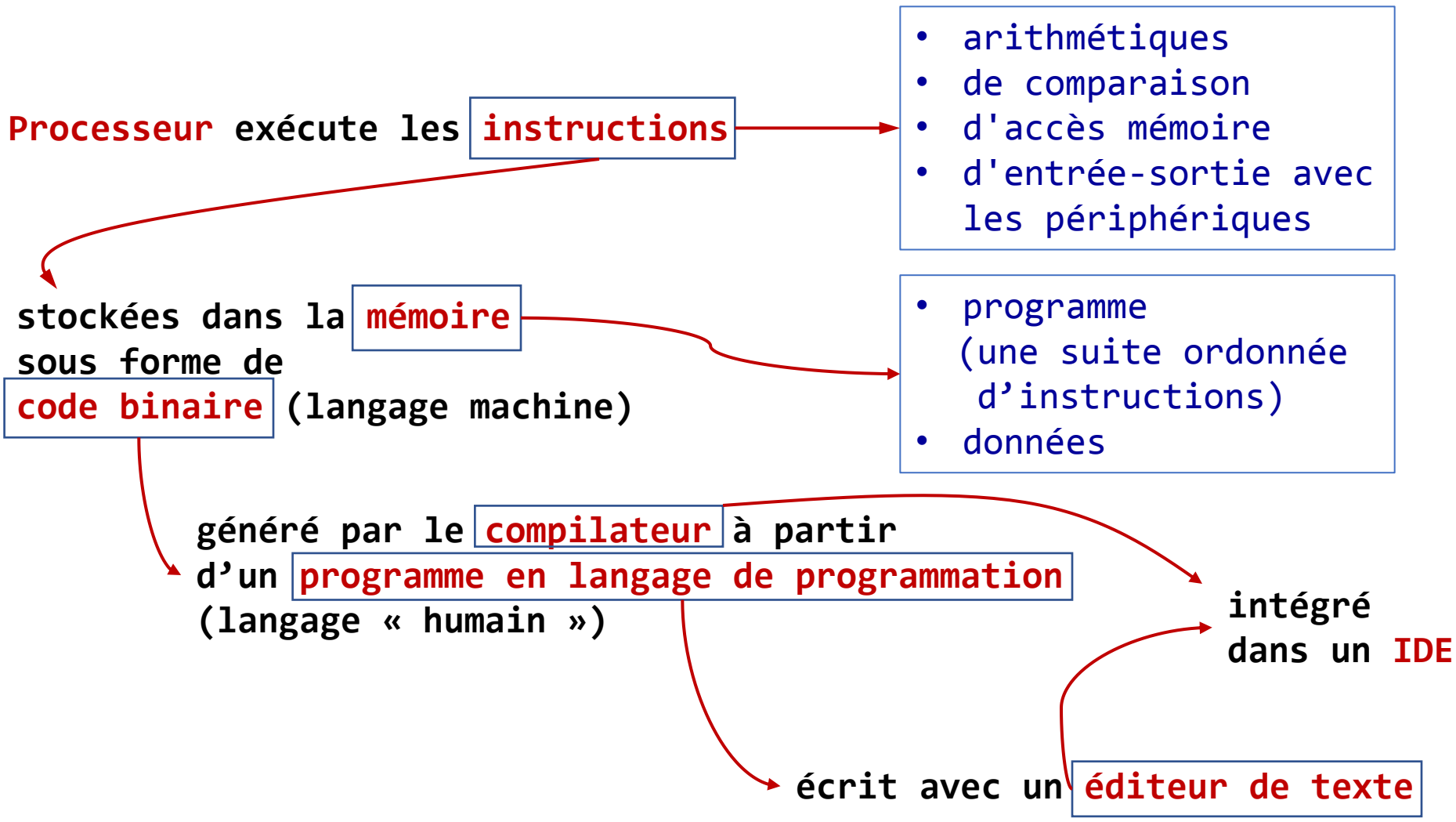
Compilateur en ligne pour pratiquer le C

onlinegdb

https://www.onlinegdb.com



Résumé


Processeur exécute les instructions
• arithmétiques
• de comparaison
• d'accès mémoire
• d'entrée-sortie avec les périphériques
stockées dans la mémoire sous forme de code binaire (langage machine)
• programme (une suite ordonnée d'instructions)
• données
généré par le compilateur à partir d'un programme en langage de programmation (langage « humain »)
intégré dans un IDE
écrit avec un éditeur de texte

Exercice 01

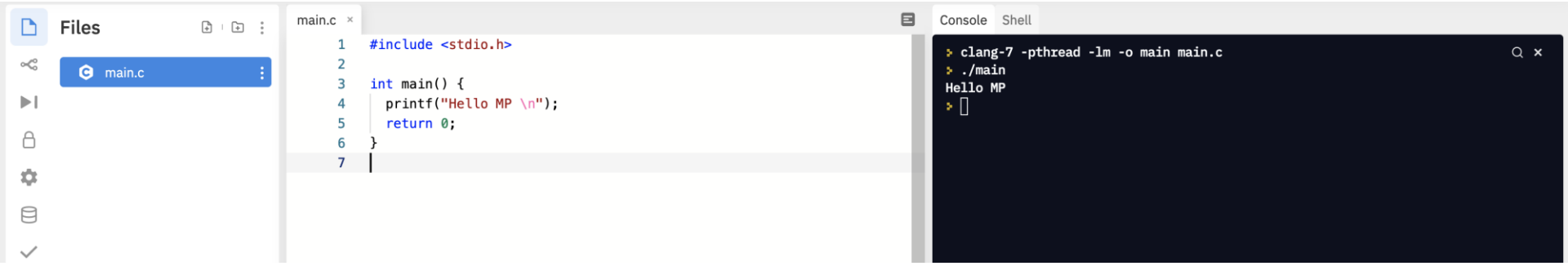
Créer un compte sur l'un des trois compilateurs enligne suivant :

- https://replit.com
- https://www.programiz.com/c-programming/online-compiler/
- https://www.onlinegdb.com

Écrire le programme suivant, puis le sauvegarder sous le nom main.c et assurer que il fonctionne bien.

```c
#include <stdio.h>

int main() {
  printf("Hello MP \n");
  return 0;
}
```

Exemple de réalisation sur replit.com



Plan

Algorithme

Une suite ordonnée d'instructions étape par étape qui indique à l'ordinateur (ou microcontrôleur) la démarche à suivre pour résoudre un problème ou une fonction.

Exprimé en :

- langage français
- schéma bloc

La programmation

Traduire l'algorithme en langage de programmation.

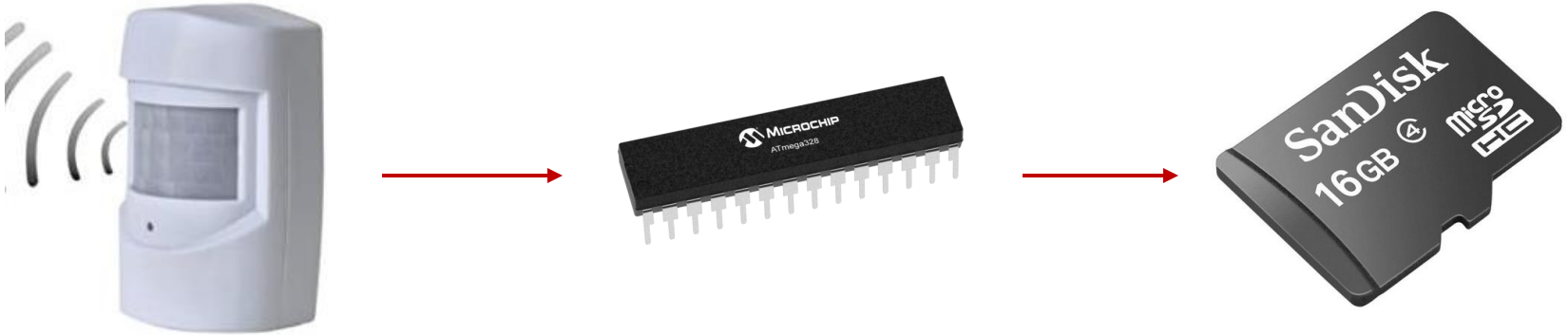
Exemple d'un problème

On veut surveiller le nombre de personnes entrant dans un magasin dans une journée pour faire des statistiques. Un ingénieur propose un système suivant :

Capteur de passage -> microcontrôleur -> carte mémoire

Notre tâche de programmeur est de programmer le microcontrôleur. Pour :

- Automatiser le comptage des personnes.

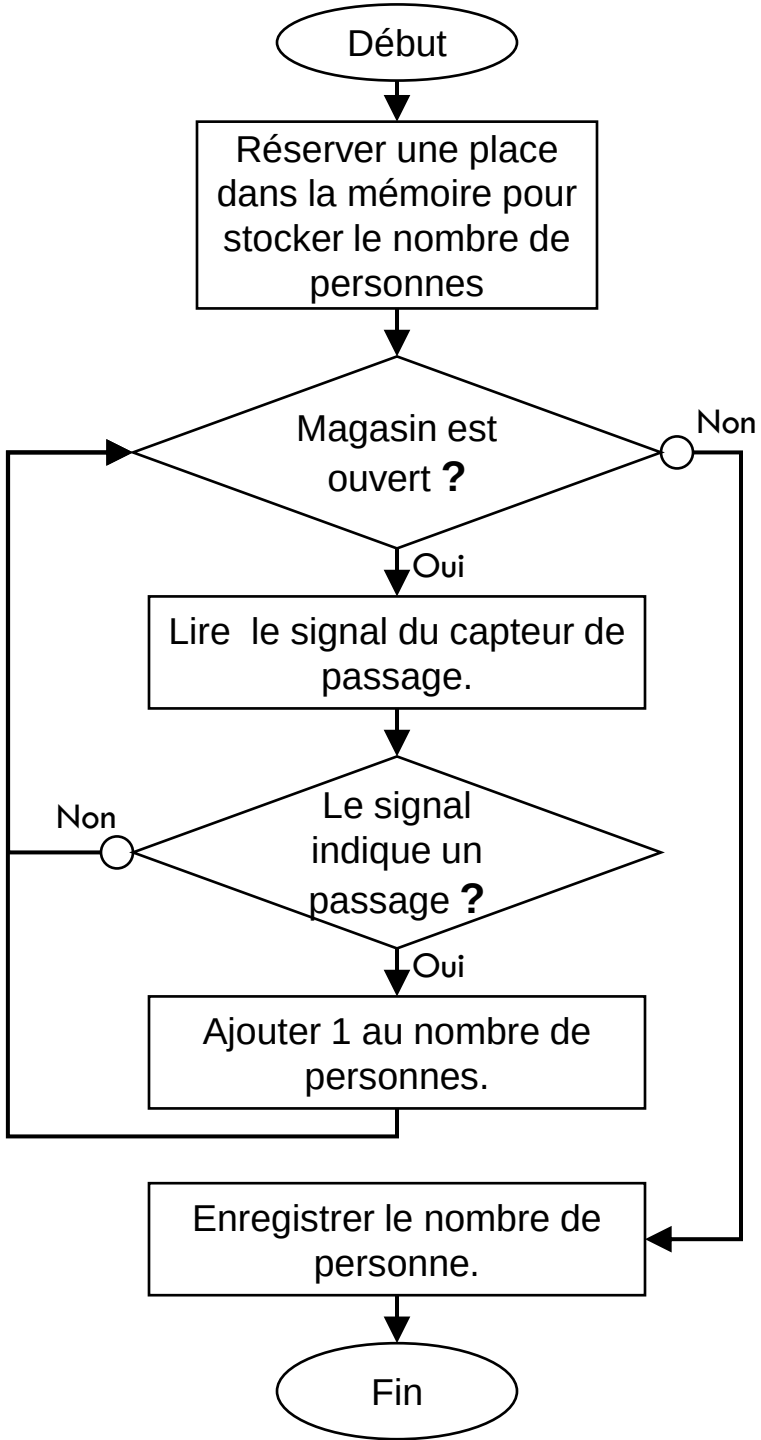


Algorithme exprimé en langage français

```
[0] Réserver une place dans la mémoire pour stocker
       le nombre de personnes
[1] Initier le nombre de personnes à zéro.
[2] Tant que le magasin est encore ouvert :
     [3] Lire le signal du capteur de passage.
     [4] Si le signal indique un passage.
          [5] Incrémenter le nombre de personne.
[6] Enregistrer le nombre de personne.
```

Algorithme exprimé en schéma bloc

Le fléchage indique l'enchaînement des étapes.



Un peu de l'histoire

1972 : l'invention du langage C par Dennis Richie et Ken Thompson, chercheurs aux Bell Labs.

1978 : la définition classique du C dans le livre "The C Programming language" de Brian Kernighan et Dennis Richie.

1989 : la définition de la norme ANSI C (puis la norme ISO en 1990)

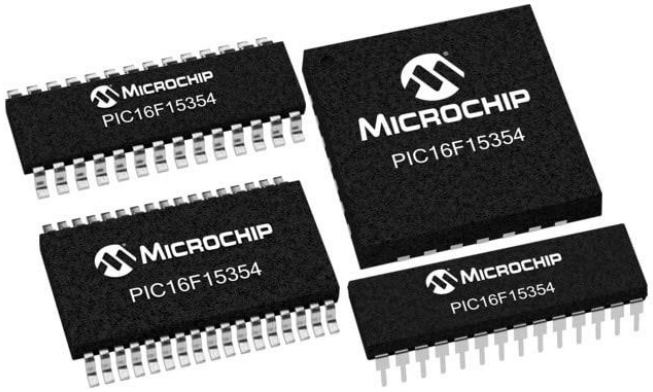
pour les systèmes embarqués

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.3
3	C		94.6
	C is used to write software where speed and flexibility is important, such as in embedded systems or high-performance computing.		
4	C++		87.0
5	JavaScript		79.5
6	R		78.6
7	Arduino		73.2

Le langage C offre également une base solide pour apprendre les autres langages.

Les microcontrôleurs
PIC de Microchip

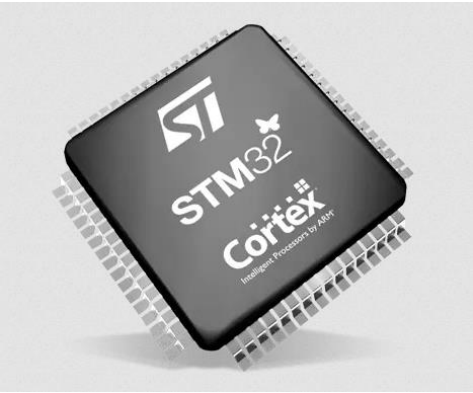


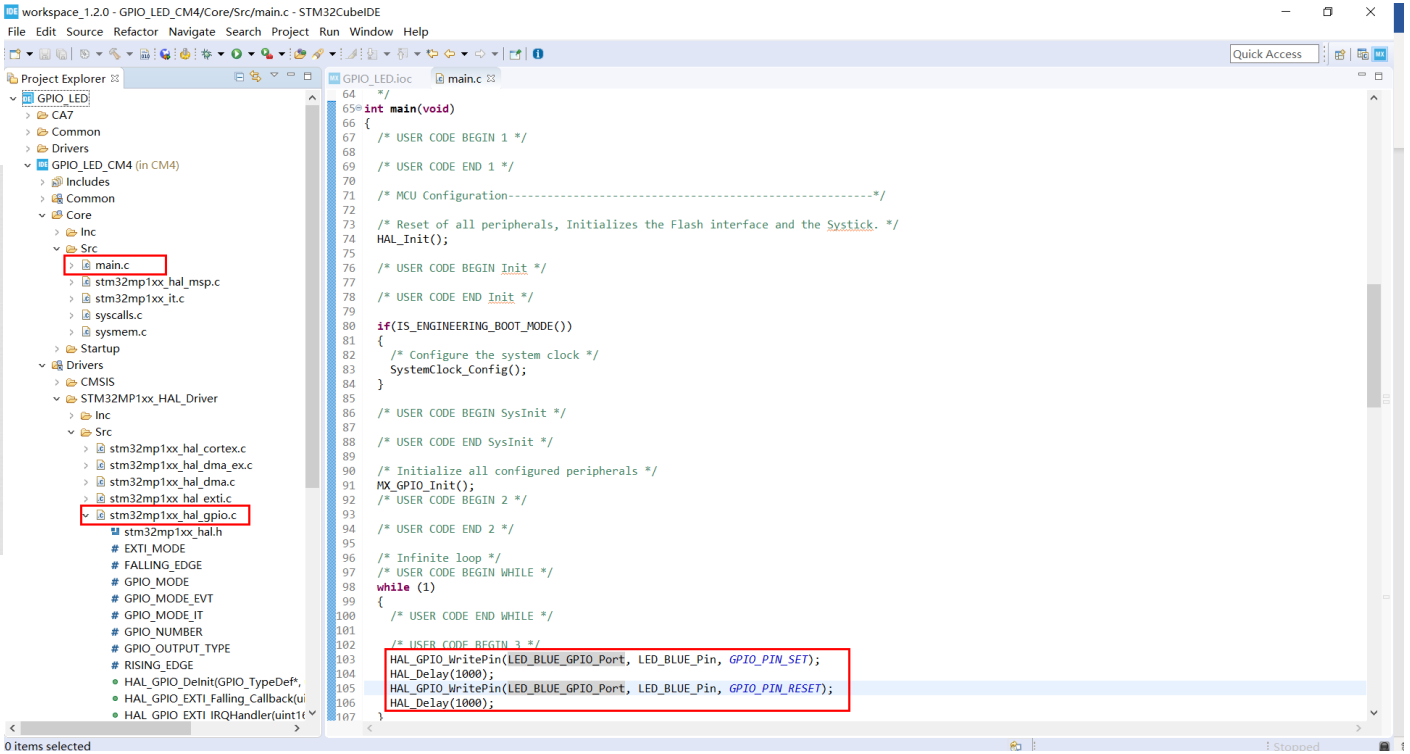
MPLAB IDE Editor

DTMFMain.c*

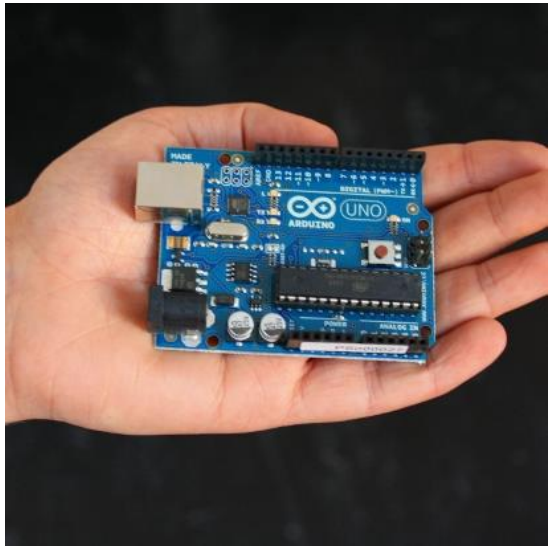
```
202   #ifdef UART_ENABLE
203     status = InDataW - InDataR;
204     if(status)
205     {
206       if(status < 0) status += INDATABUFLEN;
207
208       for(i=0;i<status;i++)
209       {
210
211         digit = InDataBuf[InDataR++];
212         if(digit == 0x2A)
213         {
214           digit = 14;
215         }
216         else if(digit == 0x23)
217         {
218           digit = 15;
219         }
220         else
221         {
222           digit -= 0x30;
223         }
224
```

Les microcontrôleurs
STM32 de STMicroelectronics





Les cartes Arduino



Blink | Arduino 1.6.13

File Edit Sketch Tools Help

Blink

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

1 Pololu A-Star 32U4 on COM4

Résumé

Algorithme = Une suite d'instructions étape par étape

présenté :
- en français
- en schéma bloc

réalisé :
- en langage de programmation

Ce qu'on va apprendre dans ce cours

Les ingrédients d'un algorithme :
- pour pourvoir construire un algorithme complet permettant de résoudre un problème.

Les syntaxes d'un langage de programmation :
- langage C
- langage LabView

Plan

Structure d'un programme C

[1] Charger les librairies nécessaires

librairie = l'ensemble des fonctions.

[2] Définir des macros

macros constantes, macros paramétrées

[3] Écrire des fonctions nécessaires qui ne sont pas disponibles dans les librairies

Optionnel mais on le fait très souvent

[4] Écrire la fonction main

Déclarer des variables

Écrire les instructions

- on réalise l'algorithme en utilisant :
  - Les opérateurs mathématiques et logiques, branchements conditionnels, boucles, tableaux et pointeurs, ...
- si besoin, on peut appeler :
  - les fonctions qu'on a écrit
  - les fonctions dans les librairies qu'on a chargé

Structure d'un programme C

Quand on écrit un programme en langage C (comme la plupart des langages de programmation), il faut comprendre qu'*on est en train d'écrire les instructions à l'ordinateur* (ou au microcontrôleur).

L'ordinateur va exécuter ces instructions
à la lettre,
les unes après les autres,
ligne après ligne.

L'ordre des instructions est *extrêmement important*. Ce qui doit être exécuté avant doit être écrit avant.

On écrit donc un programme C *ligne par ligne* en respectant l'algorithme conçu.

```
int x, y, z;
x = y + z;
y = 2;
z = x;
```

est **totalement** différent de

```
int x, y, z;
y = 2;
x = y + z;
z = x;
```

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie I

- Les variables / les constantes
- Fonctions *scanf* et *printf*
- Opérateurs : = + - * / % < > == …
- Branchements conditionnels : if, if…else, switch…case
- Boucles de répétition : while, do…while, for

Partie II

- Écrire les fonctions en C
- Tableaux à une dimension
- Pointeurs
- Lire un ficher texte de données
- Écrire un ficher texte de données
- Les macros

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie I

- Les variables / les constantes
- Fonctions *scanf* et *printf*
- Opérateurs : = + - * / % < > == …
- Branchements conditionnels : if, if…else, switch…case
- Boucles de répétition : while, do…while, for

Ingrédients d'un algorithme

Les variables

- Les variables sont utilisées pour **stocker** des nombres ou d'autres informations.
- Avant toute utilisation, il faut **absolument** les déclarer.
- La déclaration est équivalente à dire à l'ordinateur de **réserver une place dans sa mémoire** pour stocker la valeur de cette variable.
- Alors que la **place dans la mémoire** est fixe (avec une adresse bien déterminée dans la mémoire), son **contenu** peut être modifié.

Traduction en C

Template :

```
<son_type>  <son_nom>;
```

Exemple :

```
int nombre_personne;
float tension_capteur;
char une_caractere;
```

0	1	0	1	0	0	0	0	1
1	1	1	1	1	1	0	0	0
2	1	0	1	0	0	0	0	1
3	0	0	0	1	1	1	1	1
4	1	1	1	1	1	0	0	0
5	1	0	1	0	0	0	0	1

Le type définit la représentation (codage) en mémoire.
Selon la machine (microcontrôleur), la représentation d'un même type peut être différente.

Exemple du ATMega328 intégré dans les cartes Arduino Uno

```
char   -- 8  bits -- pour stocker des caractères
int    -- 16 bits -- pour stocker des nombres entiers
float  -- 32 bits -- pour stocker des nombres réels
                     (les nombres à virgule flottante simple précision)

double -- 64 bits -- pour stocker des nombres réels
                     (les nombres à virgule flottante double précision)

unsigned int -- 16 bits -- pour stocker des nombres entiers positifs
```

0	1	0	1	0	0	0	0	1	→ "a"
1	1	1	1	1	1	0	0	0	→ "14"
2	1	0	1	0	0	0	0	1	
3	0	0	0	1	1	1	1	1	

Ingrédients d'un algorithme

Les variables

Exemple

```
int x;      // dire à l'ordinateur de réserver une case de 16 bits
                dans sa mémoire
                pour stocker une variable de valeur entière qu'on va appeler x

x = 1;      // dire à l'ordinateur de mettre 1 à cette case de mémoire

x = x + 2;  // dire à l'ordinateur d'aller à l'endroit où on a stocké x,
                puis récupérer sa valeur,
                puis additionner 2 à cette valeur,
                puis remettre le résultat à l'endroit où on a stocké x.
            // après avoir exécuté la 3è lignes, x égale à 3

x = lire_capteur(); // bien sûr qu'on peut faire autre chose avec x
```

Les variables, c'est à nous la liberté de les déclarer et les utiliser.

Avant toute chose, il faut réfléchir sur le choix du type de la variable à déclarer.

Ingrédients d'un algorithme

Les constantes

C'est des valeurs qu'on donne **explicitement** à une variable dans le programme. On a le droit de faire ça.
Le type de la constante est déterminé automatiquement par le compilateur selon la façon dont la constante est écrite.

Une **constante entière** :
123 0 10

```
int x;
x = 123;
```

Une **constante caractère** :
'A' 'X' '$'

```
char y;
y = 'A';
```

Une **constante réelle** peut être données :

- en notation décimale :
  12.05 -0.01 5.0
- en notation scientifique :
  120.5e-1 -1E-4 1.5E2

```
float z;
z = 12.05;
```

Ingrédients d'un algorithme

Les constantes

Une constante peut également être donné par sa représentation hexadécimale ou binaire.

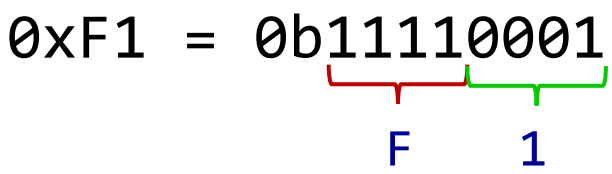
Représentation **hexadécimale** (avec préfixe **0x**)

```
0xFF01      0x54
```

Représentation **binaire** (avec préfixe **0b**)

```
0b100111    0b10
```

Attention : un nombre hexadécimale est codé sur 4 bits.



Convertisseur décimal, hexadécimal, binaire (en ligne) :
https://sebastienguillon.com/test/javascript/convertisseur.html

Ingrédients d'un algorithme

La fonction scanf

On peut également donner une valeur à une variable **via le clavier** de l'ordinateur à l'aide de la fonction **scanf**.

La fonction **scanf** appartient à la librairie standard des entrées/sorties du langage C qui s'appelle **<stdio.h>**

→ pour pouvoir l'utiliser, il faudra charger la librairie **<stdio.h>** au début du programme.
(revoir l'étape 1 de la Structure d'un programme C)

La fonction elle-même fait plusieurs choses :

- Elle attend l'utilisateur saisir une valeur : un nombre, un caractère, une chaîne de caractères.
- Elle récupère ensuite ce nombre et le mettre dans la variable indiquée.

Cette fonction permet une interaction entre l'utilisateur et le programme.

C'est une fonction qu'on va beaucoup utiliser pour pratiquer le C.
Il faut absolument savoir l'utiliser.

Ingrédients d'un algorithme

La fonction scanf

La fonction a besoin de connaître **2** choses :

- Le **type** de la valeur à récupérer : réel, entier, caractère, …
- Le **nom** de la variable pour stocker cette valeur.
  - ou plus précisément l'adresse de la variable pour stocker la valeur saisie.

```
scanf("%d", &x);
```

Type	Lettre
int	%d
float	%f
double	%lf
char	%c
string (char*)	%s

string = une chaîne de caractères

Variable : x

Opérateur & permet de trouver l'adresse de x

Ingrédients d'un algorithme

La fonction printf

On peut **afficher** la valeur d'une variable sur l'**écran** de l'ordinateur à l'aide de la fonction **printf**.

La fonction printf appartient à la librairie **<stdio.h>**.
→ pour pouvoir l'utiliser, il faudra charger la librairie **<stdio.h>** au début du programme.

C'est une fonction qu'on va beaucoup utiliser pour pratiquer le C.
Il faut absolument savoir l'utiliser.

Ingrédients d'un algorithme

La fonction printf

Étape 1 : Concevoir le message à afficher

"Je veux afficher ☐ puis ☐ puis ☐ . Voilà."

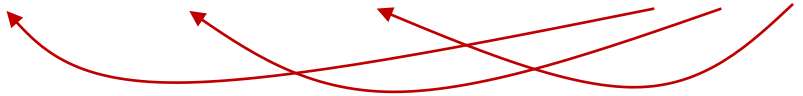
x y z

Étape 2 : Identifier le type des variables à afficher dans le message.

x est de type **int**
y est de type **char**
z est de type **float**

Étape 3 : Écrire la fonction **printf** correspondante.

```
printf("Je veux afficher %d puis %c puis %f. Voilà.", x, y, z);
```



Type	Lettre
int	%d
float	%f
double	%lf
char	%c
string (char*)	%s

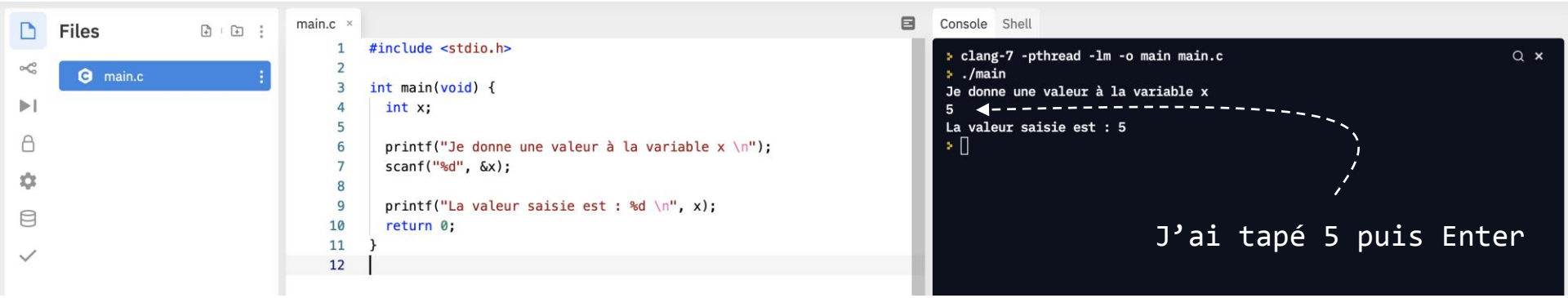
string = une chaîne de caractères

Exercice 02

1. Écrire ce programme et tester son fonctionnement.

```c
#include <stdio.h>

int main(void) {
  int x;

  printf("Je donne une valeur à la variable x \n");
  scanf("%d", &x);

  printf("La valeur saisie est : %d \n", x);
  return 0;
}
```

Exemple de réalisation sur replit.com



Exercice 02

2. Modifier le programme précédent en changeant le type de la variable x puis tester son fonctionnement.

On essayera les trois cas suivant :

- x est un float
- x est un double
- x est un char

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie I

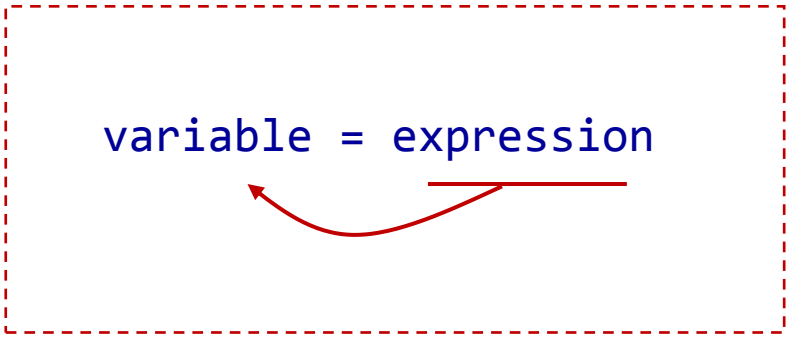
- Les variables / les constantes
- Fonctions *scanf* et *printf*
- Opérateurs : = + - * / % < > == …
- Branchements conditionnels : if, if…else, switch…case
- Boucles de répétition : while, do…while, for

Ingrédients d'un algorithme

Opérateurs

Affectation =

Le symbole "=" dans les langages de programmation
n'est pas une comparaison.



expression peut être:

- une autre variable
- une constante
- une fonction

Ça veut dire mettre la valeur de l'expression dans la variable.
L'ordre d'**affectation** est **de droite à gauche**.

Attention, inverser l'ordre, c-à-d écrire : expression = variable, a une autre signification, et est à éviter.

Ingrédients d'un algorithme

Opérateurs

Opérateurs arithmétiques

+	addition
-	soustraction
*	multiplication
/	division
%	(modulo) calcul le reste de la division entre deux nombres entiers

Quand on écrit :

```
x = a / b;
```

au niveau de l'ordinateur, ça se passe en 2 étapes :

- il calcule d'abord a/b
- puis il met le résultat de cette opération dans x
  (dans l'emplacement dans la mémoire qu'il a réserver pour x)

Attention aux cas particuliers.

une division entre deux valeurs de type **int**

- le résultat de 1/5 est 0
- le résultat de 6/5 est 1

pour que l'ordinateur donne les résultats réels corrects,
on peut écrire 1.0/5 ou 1/5.0

Exercice 03

Réaliser le programme suivant, puis tester avec différentes valeurs des variables, comparer les résultats avec un calcul manuel à l'aide d'une calculatrice.

main.c

```c
#include <stdio.h>

int main(void) {

  int a;
  int b;
  float c;
  int d;

  printf("Donnez une valeur de a \n");
  scanf("%d", &a);
  printf("Donnez une valeur de b \n");
  scanf("%d", &b);

  c = a/2.5;
  d = a%b;

  printf("La valeur de c est : %f \n", c);
  printf("La valeur de d est : %d \n", d);

  return 0;
}
```

J'ai tapé 7 puis Enter

J'ai tapé 3 puis Enter

Console Shell

```
clang-7 -pthread -lm -o main main.c
./main
Donnez une valeur de a
7
Donnez une valeur de b
3
La valeur de c est : 2.800000
La valeur de d est : 1
```

Ingrédients d'un algorithme

Opérateurs

Opérateurs relationnels (de comparaison)

`>`	strictement supérieur à
`>=`	supérieur ou égal à
`<`	strictement inférieur à
`<=`	inférieur ou égal à
`==`	égal à
`!=`	différent de

Ces opérateurs retournent une valeur 0 ou 1.

- Cette valeur vaut **1** si la condition est **vraie**
- et **0** si la condition est **fausse**.

Exemple :

```
char x;
x = 3 > 5;
```

→ la valeur de x est 0

Comme le type booléen n'existe pas en C, on utilisera le type **char**.

Ingrédients d'un algorithme

Opérateurs

Opérateurs logiques

```
&&      ET logique
||      OU logique
!       négation logique
```

Ces opérateurs retournent une valeur **0** ou **1**.
Ils sont appliqués aux expression de type relationnel.

Exemple :

```
x = (a > b) && (a < c);
```

La valeur de x sera égale à :

- 1 si a est supérieur à b ET a est inférieur de c
- 0 sinon (c-à-d, si a est inférieur ou égal à b OU a est supérieur ou égal à c)

Attention, il faut éviter à écrire des expressions logiques combinées comme ça :

```
x = b < a < c
```

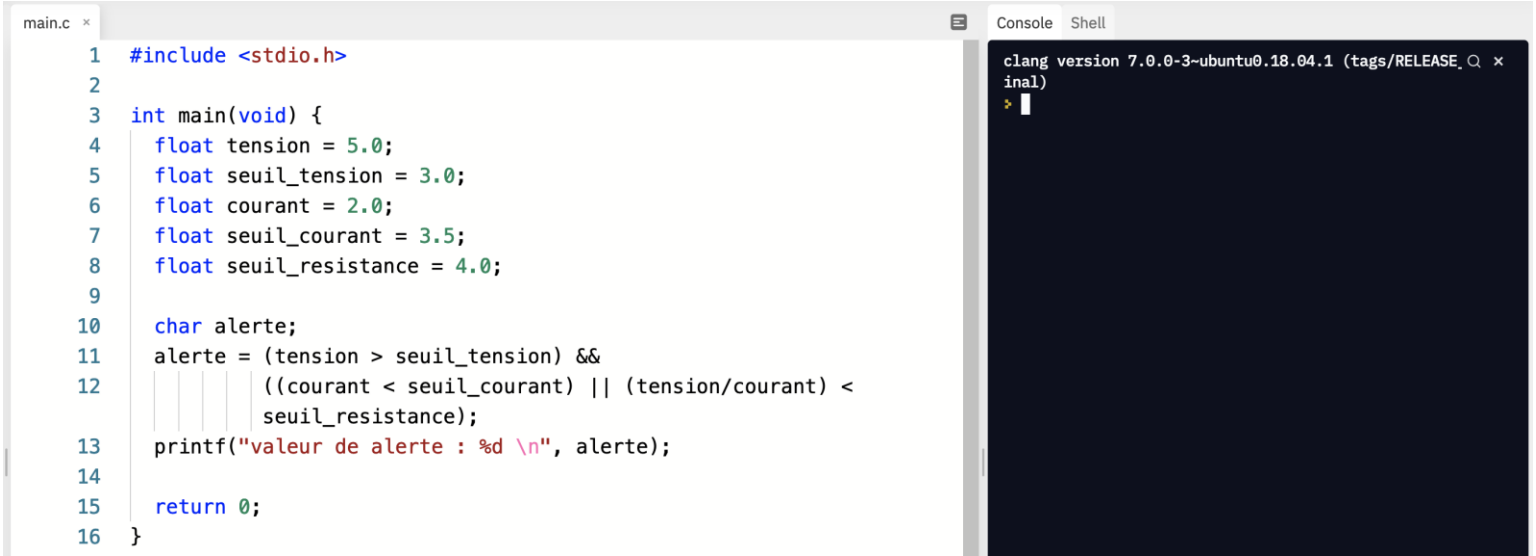
Il faut la *couper en deux* : `x = (b < a) && (a < c)`

Exercice 04

Après le calcul, quelle est la valeur de la variable alerte ?

```
tension = 5.0;
seuil_tension = 3.0;
courant = 2.0;
seuil_courant = 3.5;
seuil_resistance = 4.0;

alerte = (tension > seuil_tension) &&
         ((courant < seuil_courant) || (tension/courant) < seuil_resistance);
```

Faire un programme pour vérifier votre calcul.



Opérateurs conditionnel "?"

Template :

```
condition ? expression_1 : expression_2
```

Si la condition est vraie, expression_1 sera exécutée.
Si la condition est fausse, expression_2 sera exécutée.

Exemple, on veut obtenir la valeur absolue de x et mettre dans y.

```
y = x < 0 ? -x : x;
```

condition

expression_1

expression_2

Opérateurs logiques bit à bit

Ces opérateurs permettent de **manipuler** la **représentation binaire** des valeurs numériques.

`&`	ET
`|`	OU inclusif
`^`	OU exclusif
`~`	NON
`<<`	décalage à gauche
`>>`	décalage à droite

```
char x, y, z;
x = 0b10001000;
y = 0b11110000;
z = x & y;      // → z = 10000000
z = x | y;      // → z = 11111000
z = x ^ y;      // → z = 01111000
z = ~x;         // → z = 01110111
```

```
// décaler la représentation de x à gauche de 2 bits
z = x << 2;     // → z = 00100000
```

On remarque l'ajout de 2 bits 0 à la fin.

```
// décaler la représentation de y à droite de 3 bits
z = y >> 3;     // → z = 11111110
```

On remarque l'ajout de 3 bits 1 au début.

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie I

- Les variables / les constantes
- Fonctions *scanf* et *printf*
- Opérateurs : = + - * / % < > == …
- Branchements conditionnels : if, if…else, switch…case
- Boucles de répétition : while, do…while, for

Ingrédients d'un algorithme

Branchements conditionnels

if

Un branchement avec **if** permet de faire un **test** puis exécuter une **tâche si et seulement si** le **test** est **vrai**.

Template :

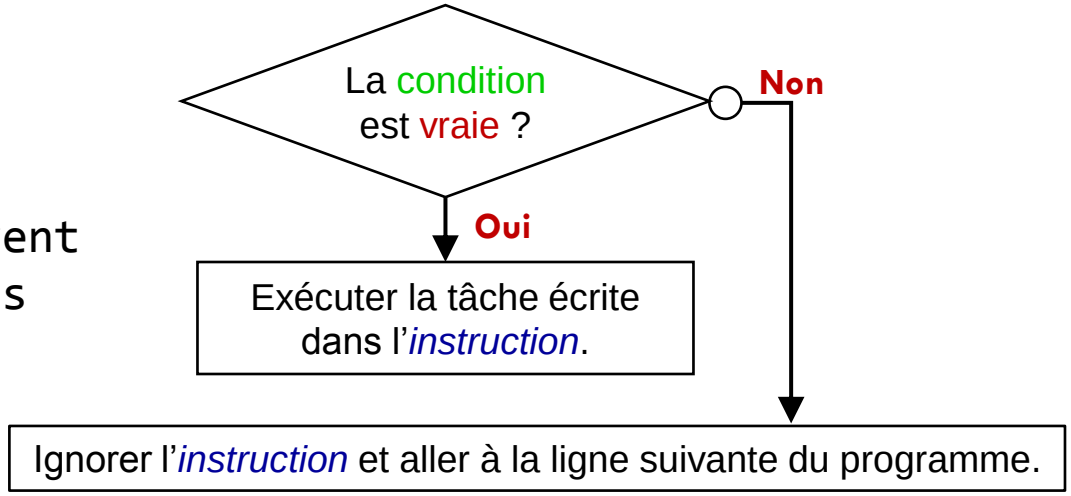
```
if (condition)
{
    instruction;
}
```

le test

la tâche

Si la condition est vraie, *instruction* sera exécutée.
Si la condition est fausse, *instruction* ne sera pas exécutée.

Dans l'expression de la condition, on utilise souvent les opérateurs relationnels (de comparaison).



Ingrédients d'un algorithme

Branchements conditionnels

if…else

Un branchement avec **if…else** permet de faire la même chose que **if** mais gère également le cas où le test est fausse.

Template :

```
if (condition)
{
    instruction_1;
}
else
{
    instruction_2;
}
```

Si la condition est vraie, seule *instruction_1* sera exécutée.
Si la condition est fausse, seule *instruction_2* sera exécutée.

Exemple avec if :

```
if (x > 0)
    {
        printf("x est positif");
    }
```

Exemple avec if…else :

```
if (x > 0)
    {
        printf("x est positif");
    }
else
    {
        printf("x est négatif ou nul");
    }
```

Exercice 05

Tester ce programme en donnant différentes valeurs pour la variable note.

main.c

```c
#include <stdio.h>

int main(void) {

  int note;

  printf("Donnez une valeur pour note : ");
  scanf("%d", &note);

  if (note >= 12)
  {
    printf("La note est bonne. \n");
  }
  else
  {
    if (note < 12 && note >= 9)
    {
      printf("La note est plutôt moyenne. \n");
    }
    else
    {
      printf("La note est faible. \n");
    }
  }

  return 0;
}
```

Console Shell

```
clang-7 -pthread -lm -o main main.c
./main
Donnez une valeur pour note : 16
La note est bonne.
```

Ingrédients d'un algorithme

Branchements conditionnels

switch…case

Un branchement avec **switch…case** permet de faire la même chose que **if…else** mais organiser les choix par catalogues (case en anglais).

```
Template :
    switch (expression)
    {
        case constante_1:
            instruction_1;
            break;
        case constante_2:
            instruction_2;
            break;
        default:
            instructions par défaut;
            break;
    }
```

Si la valeur de l'**expression** est égale à **constante_1**, alors seule ***instruction_1*** sera exécutée.

Si la valeur de l'**expression** est égale à **constante_2**, alors seule ***instruction_2*** sera exécutée.

Si ni l'un ni l'autre, alors l'***instructions par défaut*** *sera exécutée.*

On peut avoir autant de "**case**" qu'on veut.
Il faut **absolument** respecter cette structure avec les accolades **{}**, **break;** …

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie I

- Les variables / les constantes
- Fonctions *scanf* et *printf*
- Opérateurs : = + - * / % < > == …
- Branchements conditionnels : if, if…else, switch…case
- Boucles de répétition : while, do…while, for

Les **boucles** en générale permettent de **répéter une série d'instructions** tant qu'une certaine condition reste encore vraie.

3 types de boucles très utilisés :

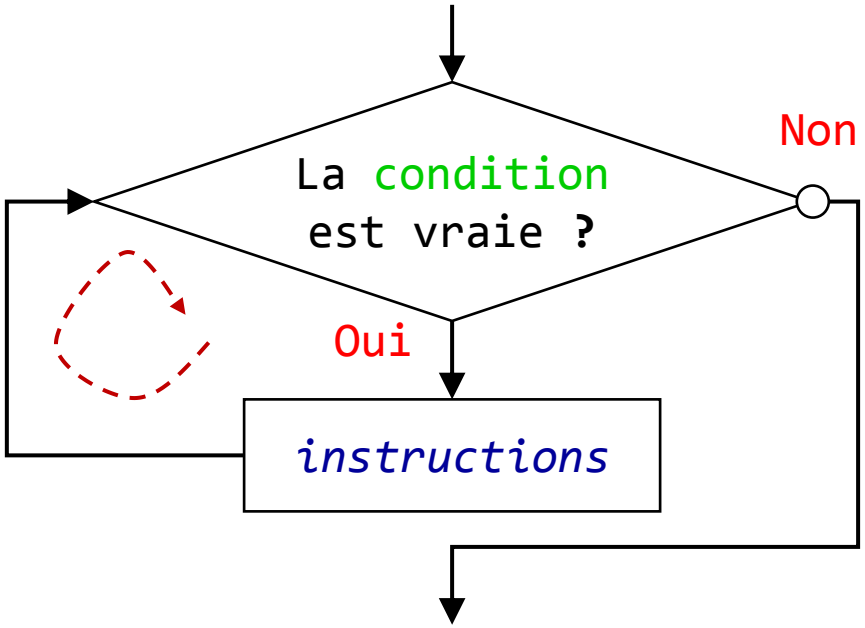
- while (tant que… faire)
- do…while (faire… tant que)
- for (pour)

Les boucles de répétition paramétrées sont indispensables pour tous les algorithmes.
Il faut absolument maîtriser leur utilisation.

```
Template :
    while (condition)
    {
        instructions;
    }
```

Tant que la **condition** reste encore vraie, les *instructions* (entre les accolades {}) seront exécutées.

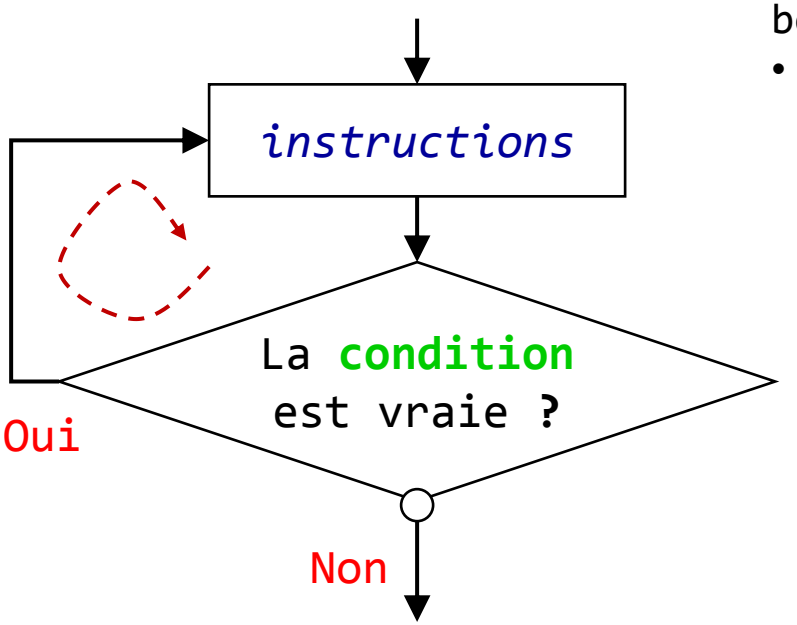
Pour quoi c'est une **boucle** ?



```
Template :
    do
    {
        instructions;
    }
    while (condition);
```

On exécute les *instructions* puis tant que la **condition** reste encore vraie, l'exécution des *instructions* sera répétée.

Pour quoi c'est une **boucle** ?



Une différence avec une boucle while :

- Quoi qu'il en soit, *instructions* seront exécutées au moins une fois.

Ingrédients d'un algorithme

Boucles de répétition

for

```
Template :
    for(initialisation; condition; incrémentation)
    {
        instructions;
    }
```

initialisation : permet d'initier le compteur de boucle
condition : permet de définir la valeur finale du compteur (où on arrête.)
incrémentation : permet de faire évoluer le compteur de boucle.

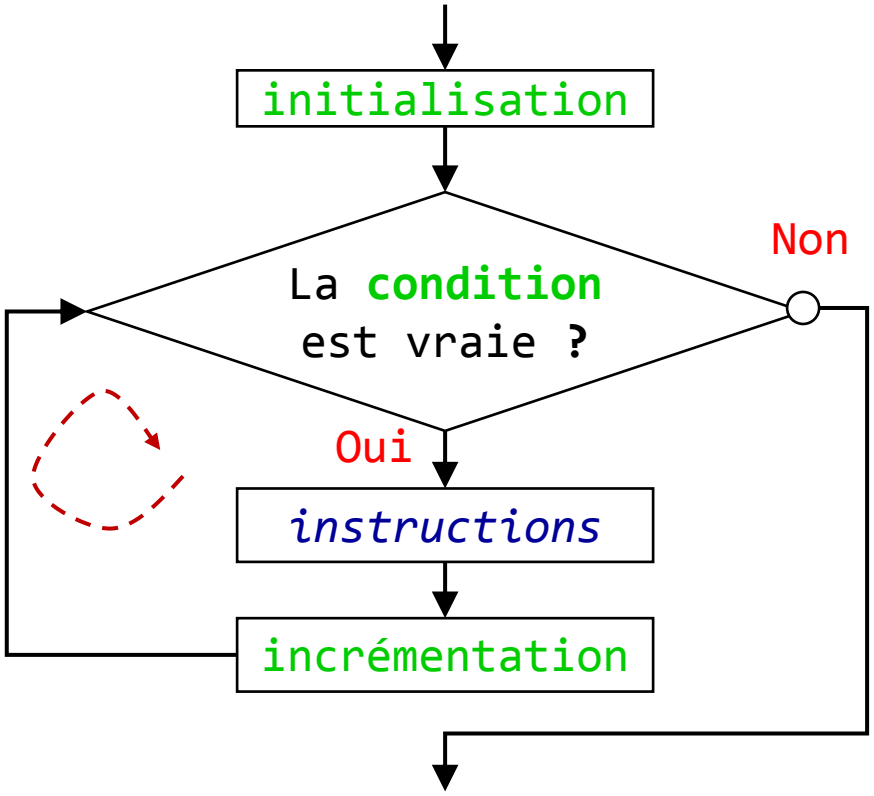
A chaque rebouclage, on incrémente le compteur et tant que la valeur du compteur ne dépasse pas la valeur finale, l'exécution des instructions sera répétée.

Ingrédients d'un algorithme

Boucles de répétition

for

```
Template :
    for(initialisation; condition; incrémentation)
    {
        instructions;
    }
```



Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie II

- Écrire les fonctions en C
- Tableaux à une dimension
- Pointeurs
- Lire un ficher texte de données
- Écrire un ficher texte de données
- Les macros

Écrire les fonctions en C

Introduction

On a vu les fonctions comme *scanf*, *printf*.

- Ces fonctions elles-mêmes sont réalisées par plusieurs lignes de code.
  En tant qu'utilisateur, on n'a pas besoin de savoir comment elle sont écrites, mais on s'intéresse plutôt à leur utilité et les syntaxes qui permet de les utiliser correctement.
  Et c'est très pratique comme ça.

- Elle sont déjà écrites par des autres personnes et disponibles dans les librairies standards du langage C.

- Elle sont partagées à tout le monde.

- Il existe beaucoup de fonctions et librairies comme ça.

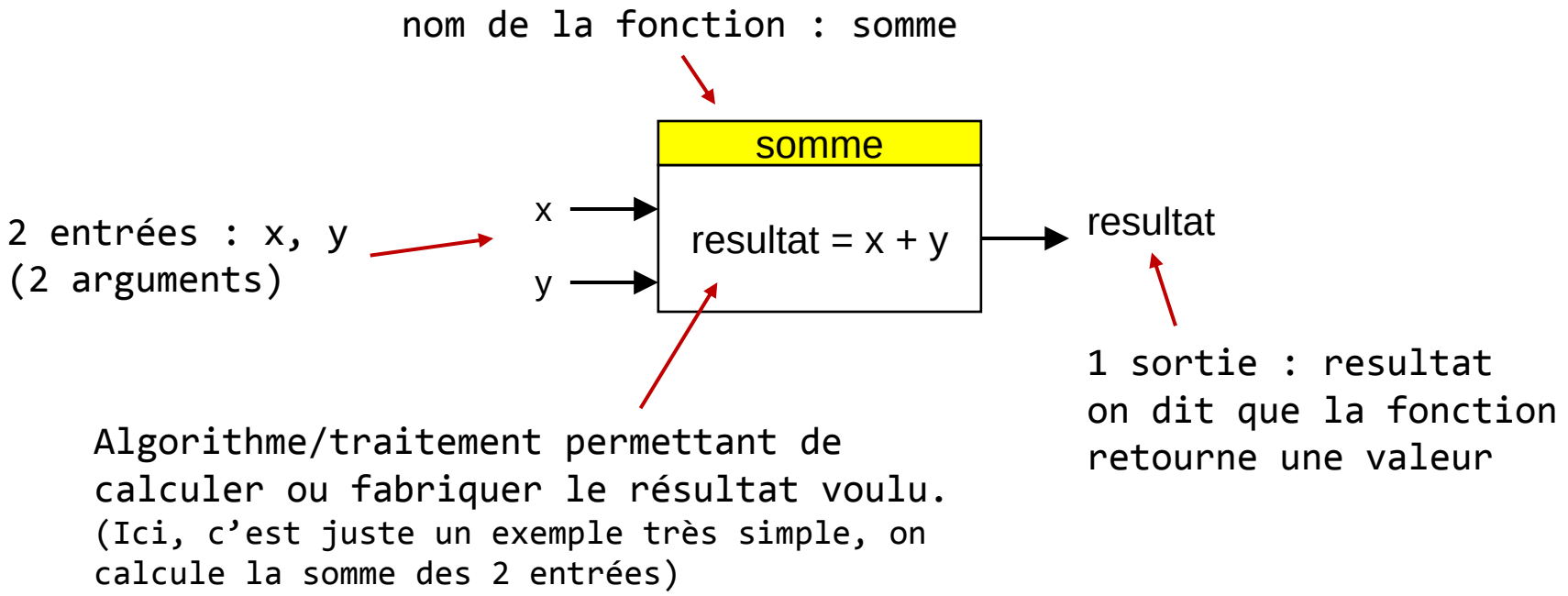
En C, on peut **écrire des nouvelles fonctions** qui peuvent utiliser celles existantes.

Ingrédients d'un algorithme

Écrire les fonctions en C

Conception

Une fonction permet de faire une tâche qu'on peut réutiliser n'importe où et à l'importe quel moment.

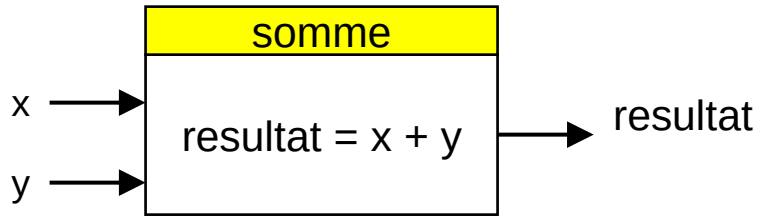


Comment utiliser cette fonction :

```
z     = somme(3,5);
total = somme(z,8);
```

```
Template :
    type_de_la_sortie    nom_de_la_fonction (déclaration_des_entrées)
    {
        instructions;
        return la_sortie;
    }
```

```
float somme(float x, float y)
{
    float resultat;
    resultat = x + y;
    return resultat;
}
```



Ingrédients d'un algorithme

Écrire les fonctions en C

La fonction principale : *main*

La *fonction principale main* se distingue des autres fonctions :

- C'est la seule fonction qui est exécutée lors de l'appel du programme.
- Les autres fonctions, qu'on va appeler les *fonctions routines*, seront exécutées si et seulement si elles sont appelées dans la fonction *main*.
- La définition des fonctions routines peut être placées avant la fonction principale *main*.

Template de la fonction principale *main* :

```
int main()
{
    instruction_1;
    instruction_2;
    …
    return 0;
}
```

Ici, on écrit le programme principal où on peut appeler les fonctions routines si nécessaire.

Cette partie est fixée.
Il faut laisser telle quelle.

Ingrédients d'un algorithme

Écrire les fonctions en C

Exemple

```c
#include <stdio.h>

float somme(float x, float y)
{
    float resultat;
    resultat = x + y;
    return resultat;
}

int main()
{
    float a, b, c;
    a = 1.2;
    b = 3.4;
    c = somme(a, b);      // retourner c = a + b
    printf("c = %f", c); // imprimer c sur l'ecran

    return 0;
}
```

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie II

- Écrire les fonctions en C
- Tableaux à une dimension
- Pointeurs
- Lire un ficher texte de données
- Écrire un ficher texte de données
- Les macros

Ingrédients d'un algorithme

Tableaux à une dimension

Déclaration – Création

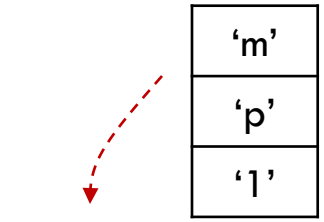
Un tableau est un ensemble des éléments de même type.
Un tableau est fait pour stocker une liste de données, par exemple : les notes des étudiants, des mesures de tension,…

Un tableau peut être pensé comme une colonne Excel ou Calc.

Template :

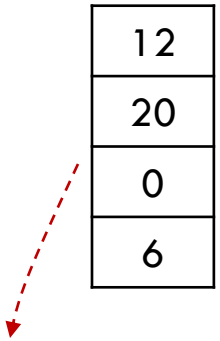
```
type_des_éléments   nom_du_tableau[nombre_de_lignes]
```

Un tableau de 3 lignes pour stocker des caractères



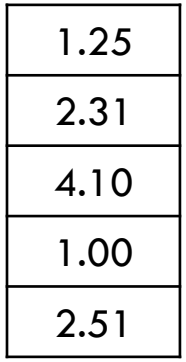

```
char monTab [3] = {'m','p','1'};
```

Un tableau de 4 lignes pour stocker des nombres entiers




```
int autreTab [4] = {12, 20, 0, 6};
```

Un tableau de 5 lignes pour stocker des nombres réels




```
float Tab [5] = {1.25, 2.31, 4.10, 1.00, 2.51};
```

Tableaux à une dimension

Accès aux éléments d'un tableau à une dimension

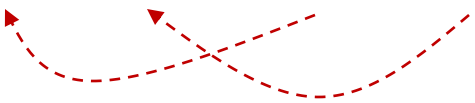
Accès aux différentes lignes d'un tableau se fait par :

nom_du_tableau[*indice_de_la_ligne*]

Attention : en C, tout commence de **0**.
C-à-d, la première ligne du tableau monTab est monTab[0].

Exemple

```c
#include <stdio.h>
int main()
{
    float monTab[3] = {1.5, 2.6, 3.7};
    int i;
    for (i = 0; i < 3; i = i+1)
    {
        printf("monTab[%d] = %f \n", i, monTab[i]);
    }
    return 0;
}
```



Passer un tableau à une dimension à une fonction

Pour pouvoir passer un tableau à une fonction, il faut 2 choses :

1. Lors de la *définition* de la fonction, préciser dans la **liste des entrées** que la fonction peut recevoir un tableau à une dimension.
2. Lors de *l'appel de la fonction*, passer à la fonction **le nom du tableau**.

```c
#include <stdio.h>

void affiche(int tab[], int N)
{
    int i;
    for (i = 0; i < N; i = i + 1)
    {
        printf("%d \n", tab[i]);
    }
}

int main()
{
    int tabx [3] = {2, 3, 5};
    int taby [4] = {3, 2, 1};
    affiche(tabx, 3);
    affiche(taby, 4);
    return 0;
}
```

Première entrée est un tableau des nombre entiers à une dimension.

Deuxième entrée est un nombre entier.

Passage d'un tableau à une fonction.
On donne son nom à la première entrée.

Résumé

Tableaux à une dimension

Un tableau est fait pour stocker une liste de données.

Un tableau est stockés en mémoire à des adresses contiguës.

Pour accéder aux différentes lignes d'un tableau, il faut simplement préciser l'indice de la ligne entre les crochets [].

```
nom_du_tableau[indice_de_la_ligne]
```

Ligne numéro 0	12
Ligne numéro 1	20
Ligne numéro 2	0
Ligne numéro 3	6

Pour qu'un fonction puisse recevoir un tableau à une dimension, il faut le préciser dans sa définition.

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie II

- Écrire les fonctions en C
- Tableaux à une dimension
- Pointeurs
- Lire un ficher texte de données
- Écrire un ficher texte de données
- Les macros

Ingrédients d'un algorithme

Pointeurs

Les pointeurs sont conçus pour contenir l'**adresse** d'une variable, d'un tableau, d'un ficher et même d'une fonction.

Pour qu'une fonction puisse accéder au contenu d'une variable, d'un tableau ou d'un fichier, il faut absolument lui donner un pointeur qui contient leur adresse.

C'est pour cela que l'utilisation des pointeurs est indispensable en pratique.

Déclaration d'un pointeur
Template :

```
type_de_données * nom_du_pointeur;
```

```
int * ptx;   ⟵ Le pointeur ptx peut contenir l'adresse d'une variable de type int
float * pty; ⟵ Le pointeur pty peut contenir l'adresse d'une variable de type float
```

Ingrédients d'un algorithme

Pointeurs

```c
#include <stdio.h>
int main()
{
    int x = 1;
    float y = 2.5;

    int * ptx;
    float * pty;

    ptx = &x;
    pty = &y;

    printf("%p \n", ptx);
    printf("%p \n", pty);


    printf("%d \n", *ptx);
    printf("%f \n", *pty);

    return 0;
}
```

& est l'opérateur de référencement qui retourne l'adresse mémoire d'un objet (variable, élément d'un tableau, fonction).

Ici on veut afficher le contenu des pointeurs, c-à-d une adresse.
On remarque l'utilisation de "%p" puisque ptx et pty ne sont pas de type int ni de type float mais des pointeurs.

Ici on veut afficher la valeur des variable à laquelle pointent les pointeurs.
* est l'opérateur de dé-référencement qui retourne la valeur pointée par les pointeurs.

Fonctions qui reçoivent les pointeurs comme entrées

Pour qu'une fonction puisse accéder au contenu d'une variable, d'un tableau ou d'un fichier, il faut absolument lui donner un pointeur qui contient leur adresse.
Tout se passe lors de la définition de la fonction où on déclare la **liste des entrées**.

Analysez les 2 fonctions *augmenter* suivantes !

```c
#include <stdio.h>

void augmenter(int x)
{
    x = x + 1;
}

int main()
{
    int x = 5;
    augmenter(x);
    printf("%d", x);
    return 0;
}
```

```c
#include <stdio.h>

void augmenter(int * x)
{
    *x = *x + 1;
}

int main()
{
    int x = 5;
    augmenter(&x);
    printf("%d", x);
    return 0;
}
```

Plan

1. Introduction
2. Algorithme et langage C
3. Structure d'un programme C
4. **Ingrédients d'un algorithme**

Partie II

- Écrire les fonctions en C
- Tableaux à une dimension
- Pointeurs
- Lire un ficher texte de données
- Écrire un ficher texte de données
- Les macros

Ingrédients d'un algorithme

Lire un ficher texte de données

```c
#include <stdio.h>
int main()
{
     char NomFichier[] = "fichier_mesure.txt";
     FILE * PointeurFichier ;
     PointeurFichier = fopen(NomFichier, "r");

     int nombre_mesures = 6;
     float X[nombre_mesures];

     float valeur;
     int i;
     for (i = 0; i < nombre_mesures; i = i+1)
     {
          fscanf(PointeurFichier,"%f", &valeur);
          X[i] = valeur;
     }

     fclose(PointeurFichier);

     for (i = 0; i < nombre_mesures; i = i+1)
     {
          printf("x[%d] = %f \n", i, X[i]);
     }
     return 0;
}
```

1. Ouvrir le fichier texte "fichier_mesure.txt" qui est dans le même répertoire que ce programme. On remarque l'utilisation d'un pointeur de type FILE qui contient l'adresse du fichier texte pour pouvoir ensuite y accéder puis récupérer les données dans ce fichier.

2. Préparation un tableau pour stocker les données récupérées du ficher texte avec le nombre de mesures qu'on souhaite récupérer

3. On balaye chaque ligne du fichier avec une boucle for.
À chaque fois qu'on passe une ligne, on récupère le nombre trouvé dans cette ligne
- puis le mettre dans la variable valeur
- puis on remettre cette valeur dans la ligne i du tableau X

4. On ferme le fichier texte (OBBLIGATOIRE)

On écrit une boucle for pour afficher toutes les lignes du tableau pour vérifier.

Ingrédients d'un algorithme

Écrire un ficher texte de données

```c
#include <stdio.h>
int main()
{
     char NomFichier[] = "data.txt";
     FILE *PointeurFichier ;
     PointeurFichier = fopen(NomFichier, "w");



     int i, nbLigne;
     nbLigne = 20;
     for (i = 0; i < nbLigne; i = i+1)
     {
          fprintf(PointeurFichier,"%d \n", i);
     }


     fclose(PointeurFichier);

     return 0;
}
```

1. Ouvrir le fichier texte "data.txt" qui est dans le même répertoire que ce programme. On remarque l'utilisation d'un pointeur de type FILE qui contient l'adresse du fichier texte pour pouvoir ensuite y accéder puis écrire les données dans ce fichier.

2. On balaye chaque ligne du fichier avec une boucle for.
À chaque fois qu'on passe une ligne, on écrit dans chaque ligne la valeur de i

3. On ferme le fichier texte (OBBLIGATOIRE)

Plan

Ingrédients d'un algorithme

Les macros

Une ***macro*** est une abréviation de ***macro-génération de code*** qui permet une substitution de code avant la phase de compilation. C'est assez utile.

Il y a ***2 types de macros*** :

- les macros *constantes*
- les macros *paramétrées*.

En générale, les macros constantes permettent *définir une constante* qu'on va utiliser plusieurs fois dans le programme.

En générale, les macros paramétrées est, une méthode rapide pour *définir une fonction* ***courte***.

Ingrédients d'un algorithme

Les macros

On définit les macros **en début du programme** à l'aide d'une instruction préprocesseur #define.

Les macros *constantes*

Template :

```
#define  nom_de_la_constante  sa_valeur
```

```
#include <stdio.h>

#define tension_max 5

int main(void)
{
     float tension_mesuree;
     tension_mesuree = 5.5;

     if (tension_mesuree > tension_max)
     {
          printf("problème sur-tension \n");
     }
     return 0;
}
```

Après avoir défini tension_max dans une macro, on l'utilise comme si c'est une variable qui a une valeur constante de 5.

Ingrédients d'un algorithme

Les macros

Les macros *paramétrées* permet de définir les fonctions courtes d'une manière élégante.

Template :

```
#define  nom_de_la_fonction(liste_des_paramètres) (instructions_avec_les_paramètres)
```

```c
#include <stdio.h>

#define valeur_abs(x) ( x>0 ? x : -x)

int main() {
     float m;
     m = -2.5;
     printf("%f", valeur_abs(m));
     return 0;
}
```

Après avoir défini valeur_abs(x) dans une macro, on l'utilise comme si c'est une fonction.

Références

Site web

A. Canteaut, Programmation en langage C,
https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/
F. Faber, Introduction à la programmation en ANSI-C,
https://www.ltam.lu/cours-c/prg-c.htm
OpenClassRooms, Apprenez à programmer en C !,
https://openclassrooms.com/courses/apprenez-a-programmer-en-c

IDE - Integrated Development Environment (environnement de développement)

- *ChIDE* (Windows, Linux, Mac OS X)
- *Code :: Blocks + mingw* (Windows, Linux, Mac OS X)
- *Xcode* (Mac OS X)