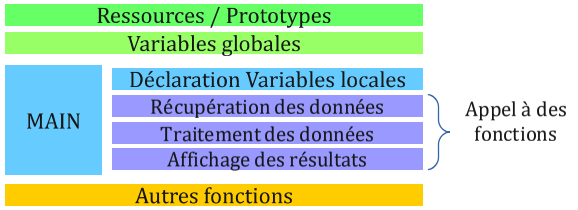
#include<stdio.h>

Langage C

Notions de C

STRUCTURE D'UN CODE C

Le langage C est un **code très strict** dans sa façon de coder.
Le **code source** doit avoir la **structure suivante** :



TYPAGE DES VARIABLES

Une variable en C est caractérisée par :

- son **adresse** : endroit de stockage en mémoire
- son **type** : nature de l'information qu'elle contient
- son **nom** : nom donnée à la variable dans le programme
- sa **valeur** : contenu de la variable

```
int k = 5;   double y = 2.45;   char c = 'g';
```

Chaque type de variable est codé sur un **certain nombre d'informations binaires** (limitant la valeur maximale qu'elle peut contenir)

ENTIERS	**int**	32 bits – 4 octets	%d
	short	16 bits - 2 octets	%d
RÉELS	**double**	64 bits – 8 octets	%lf
CARACTÈRES	**char**	8 bits – 1 octet	%c

STRUCTURE D'UN PROJET C

Afin de mieux structurer les informations et, en particulier, gérer les différentes fonctions de l'application, on peut classer ces fonctions dans des **bibliothèques.**
Chaque bibliothèque est composée :

- d'un **fichier header** (extension .h) : rassemblant les prototypes des fonctions et les constantes propres
- d'un **fichier source** (extension .c) : rassemblant les définitions des fonctions

tableau1D.h

```
#ifndef TABLEAU1D_H_INCLUDED
#define TABLEAU1D_H_INCLUDED
#include <stdlib.h>
#include <stdio.h>

void afficheTab(int tab, int dim);
void triTabCroissant(int tab, int dim);

[...]

#endif // TABLEAU1D_H_INCLUDED
```

tableau1D.c

```
#include "tableau1D.h"
void afficheTab(int tab, int dim){
        int i;
        for(i = 0; i < dim; i++)
        ...
}
void triTabCroissant(int tab, int dim){
        ...
}
[...]
```

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "tableau1D.h"
int main(void){
        ...
        afficheTab(montab, 10);
                                    ...
                                }
```

FONCTION

Une **fonction** est un **regroupement d'instructions.**
Elle est caractérisée par :

- son **identificateur** (ou nom)
- son **type de retour** : type de donnée retournée (1 seule)

Pour pouvoir exécuter sa suite d'instructions, une fonction peut avoir besoin de **paramètres d'entrée** (ou **arguments**) typés.

```
type nom(type arg1, type arg2);
```

Ceci est le prototype de la fonction.

Les fonctions doivent ensuite **être écrites en dehors** de toute autre fonction. Elles seront ensuite **appelées dans** d'autres fonctions.

```
int somme(int a, int b){
        int c = a + b;
        return c;                }
```

AFFICHAGE CONSOLE

Pour **afficher dans la console**, on utilise la fonction **printf.**

```
printf("Exemple de texte avec saut de ligne\n");
int a = 32;  printf("a =%d\n", a);     // entier
double k = 5.3;  printf("k=%2.4lf\n", k);
// réel avec 2 chiffres significatifs avant la virgule et 4 chiffres après
```

LECTURE CLAVIER

Pour **lire des données au clavier**, on utilise la fonction **scanf.**

```
int b;        scanf("%d",&b);        // entier
double m;  scanf("%lf",&m);        // réel
```

STRUCTURES DE CONTRÔLE

```
if((x==2) && (y > 5)){
    k = 1;
}
else{
    k = 0;
}
```

Exécute le premier bloc d'instructions si la condition est vraie, sinon exécute le second bloc d'instructions

```
switch(x){
    case 1 :
        k = 5;
        break;
    case 15 :
        k = 2;
        break;
    default :
        k = 0;
}
```

Exécute le bloc d'instructions selon le cas validé

```
for(m = 5; m < 120 ; m = m + 10 ){
    printf("m =%d\n", m);
}
```

Exécute le bloc d'instructions pour m allant de 5 à 120 par pas de 10

```
while(z <= 0){
    k++;
    Z += 0.1;
}
```

Exécute le bloc d'instructions tant que la condition est vraie

CONSTANTES

Il est également possible de déclarer des **constantes.**

```
#define M 10    #define PI 3.14    #define NOM "julien"
```

TABLEAU 1D

Les tableaux sont des **regroupements indexés** de N variables d'un **même type**, l'indice allant de **0 à N-1.**
Ils sont caractérisés par leur **nom** et le **nombre de variables** qu'ils peuvent stocker. La taille du tableau est fixée statiquement.

```
int notes[5] = {10, 15, 12, 08, 13};
double suite[M];                    int i;
for(i = 0; i < M ; i++){    suite[i] = 2*i; }
```

Remplissage d'un tableau d'entier (notes) et d'un tableau de réels (suite)

```
for(i = 0; i < M ; i++)
    printf("case %d = %lf \n", i, suite[i]);
```

Affichage d'un tableau de réels (suite)

POINTEURS

Un pointeur est une variable qui **peut stocker l'adresse** d'une autre variable.

```
type *p_nom ;
```

On distingue un pointeur d'une variable standard dans sa déclaration par le symbole *
A sa déclaration, un pointeur n'adresse **aucune variable.** Il retourne la valeur **NULL.**

L'adresse d'une variable standard est donnée en ajoutant le symbole **&** devant le nom de la variable.

```
int i = 3;
int *p_i = &i;      //p_i contient l'adresse de la variable i
printf("i =%d - *p_i = %d \n", i, *p_i);
```

ALLOCATION DYNAMIQUE

L'intérêt principal du langage C est de pouvoir optimiser au mieux l'utilisation de la mémoire centrale du calculateur.
Pour cela, il existe des fonctions permettant d'**allouer la mémoire de manière dynamique.**

```
double *k;                    int m = 5 ;
k = malloc(m);
*(k+1) = 2.67;
printf("case k+1 = %lf \n", k[1]);
```

Création d'un pointeur de type double nommé k
Allocation de 5 espaces mémoires de double au pointeur k
Remplissage de l'adresse pointée par k+1 (c'est à dire la case mémoire de type double suivant celle pointée par k)
Affichage de cette valeur

Langage C

Chaîne de caractères

DÉCLARATION

Une chaîne de caractères est un **regroupement de caractères.**

```
char ma_chaine[128];
```

Création d'un tableau de caractères de 128 cases

INITIALISATION

Il existe plusieurs manières d'initialiser une chaîne de caractères

```
char ma_chaine[128] = "Vive le C" ;
```

Création d'un tableau de caractères de 128 cases dont 10 seront utilisées :
9 caractères de la chaîne et le caractère de fin de chaîne '\0'

```
char ma_chaine[] = "Vive le C" ;
```

Création d'un tableau de caractères de 10 cases :
9 caractères de la chaîne et le caractère de fin de chaîne '\0'

```
char *ma_chaine = "Vive le C" ;
```

Création d'un pointeur vers des caractères / Allocation de 10 cases mémoires : 9 caractères de la chaîne et le caractère de fin de chaîne '\0'

BIBLIOTHÈQUE <STRING.H>

Un ensemble de fonctions regroupé dans la bibliothèque **<string.h>** permet de gérer les chaînes de caractères.

Longueur d'une chaîne

```
int k = strlen(ma_chaine);  // k = 9
```

Compte le nombre de caractères jusqu'au caractère '\0'

Copie d'une chaîne

```
char *ch_source = "Vive le C";  char ch_dest[40];
strcpy(ch_dest, ch_source);
```

La chaîne source peut être une chaîne constante. Ce qui permet d'initialiser.

Comparaison de chaînes

```
int id = strcpy(ch_dest, ch_source);
```

Renvoie '0' si les deux chaînes sont identiques, une autre valeur sinon.

Concaténation de chaînes

```
strcat(ch_dest, ch_source);
```

Concatène les deux chaînes de caractères et stocke le résultat dans la première

Écriture formatée dans une chaîne

```
double k = 5.3;  sprintf(ch_dest, "k=%2.4lf\n", k);
```

Écrit le texte formaté dans la chaîne ch_dest

Langage C

Gestion de fichiers

Pour pouvoir lire ou écrire des données dans un fichier, il existe un certains nombres de fonctions contenues dans la bibliothèque :
<string.h>

TYPES DE FICHIERS

Il existe deux grands types de fichiers dans lesquels on peut stocker des données :

- des **fichiers texte ASCII**, lisibles par n'importe quel lecteur de fichier texte (NotePad par exemple)
- des **fichiers binaires** (propriétaires), non lisibles par un lecteur standard, mais nécessitant la connaissance préalable du format

POINTEUR VERS UN FICHIER

Les fichiers sont traités en C sous forme d'un pointeur de type :

```
FILE *mon_fichier;
```

FERMETURE D'UN FICHIER

Lorsqu'un fichier est utilisé par un programme, le système d'exploitation vérouille ce fichier pour tout autre programme. Il est donc essentiel de refermer un fichier une fois qu'il n'est plus utilisé. Pour cela, on utilise la fonction suivante :

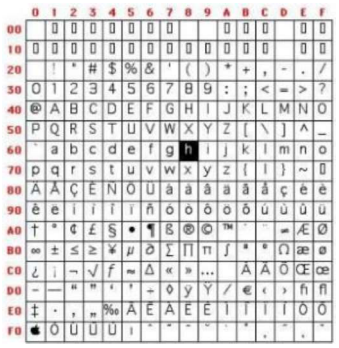
```
fclose(mon_fichier);
```

1 octet = 8 bits

2^10 = 1ki / 2^20 = 1Mi / 2^30 = 1Gi / 2^40 = 1Ti

CODE ASCII

Les variables de type **char** permettent de stocker des données sur 1 octet.
Le code **ASCII** (*American Standard Code for Information Interchange*) permet de coder l'alphabet anglais (incluant les chiffres) sur 7 bits.



```
ma_chaine[2] = 'h';
```

La case 2 de la chaine *ma_chaine* va contenir le code ASCII de la lettre h.

Il existe un **caractère spécial de fin de chaîne** : '\0'
Il permet de délimiter la fin de la chaîne.

OUVRIR UN FICHIER

Pour **ouvrir le fichier** nommé mon_fichier.txt en écriture ASCII, on utilise la fonction suivante :

```
mon_fichier = fopen("my_file.txt", "w" );
```

Cette fonction renvoie l'adresse de la première donnée récupérée du fichier, si l'ouverture s'est faite normalement.
Sinon, elle renvoie la valeur **NULL.** On peut ainsi tester l'ouverture d'un fichier par le test suivant :

```
if(mon_fichier != NULL){ /*ouverture réussie*/ }
```

Le premier paramètre est le **chemin** (ici relatif) **vers le fichier** à ouvrir (une chaîne de caractères) avec son extension.
Le second paramètre est le **mode d'ouverture** du fichier :

- **r** : *read* / lecture
- **w** : *write* / écriture (en écrasant le fichier)
- **a** : *append* / écriture (en ajoutant au fichier existant)

On peut ajouter l'option **b** (binary) pour des fichiers binaires.

```
fopen("mon_fichier.bin", "rb" );
```

Ex : pour ouvrir le fichier binaire ***mon_fichier.bin*** en lecture

ÉCRIRE DES DONNÉES / FICHIER TEXTE

L'écriture dans un fichier texte se fait par transmission de chaines de caractères, grâce à la fonction :

```
void fprintf(FILE *fic, char *format, var1, var2...)
```

Par exemple, pour écrire le texte : VARIABLE A =, suivi de la valeur de la variable a (type double par exemple), on peut utiliser l'instruction suivante :

```
fprintf(mon_fichier, "VARIABLE A =%lf \n", a);
```

LIRE DES DONNÉES / FICHIER TEXTE

La lecture dans un fichier texte se fait par récupération de chaines de caractères, grâce à la fonction :

```
void fscanf(FILE *fic, char *format, adr_var1, ...)
```

Par exemple, pour récupérer une variable entière et une chaîne de caractères dans un fichier, on peut utiliser l'instruction suivante :

```
fscanf(mon_fichier, "%d %s", a, ma_chaine);
```

Ici **a** est de type entier et **ma_chaine** est une chaîne de caractères.

Attention : la lecture d'une chaîne de caractères à l'aide de la commande ***fscanf*** est limitée au premier espace. On peut aussi utiliser la commande ***fgets*** pour contourner ce problème :

```
fgets(ma_chaine, 128, mon_fichier);
```

Le second paramètre correspond à la taille maximale de la chaîne de caractères à récupérer.

Il existe également un caractère spécifique de fin de fichier : **EOF.**