

Langage C + AISDD :

> Code Source
 (Compilateur) → Exécutable
 Débugueur

> Directives préprocesseurs.

include

> rapide, petite et coûte cher :).

> lorsque on note ordi → il faut

initialiser vos variables.

avant de faire l'initialisation, la variable a une valeur par défaut

Attention :

i++ 8 ++i
 c'est pas la même chose.
 c = ++i
 j'incrémente ~ j'affecte.

c = i++
 j'affecte ~ j'incrémente.

<math.h>

une lib qui contient les fonctions mathématiques

TIPS :

("%.f ...", i/j)
 Division réelle.

%.5.2f

pour la partie entière
 pour la partie fractionnaire

> Programme en C
 constitué de fonctions.
 console.
 fenêtré.

> mémoires

RAM ROM registres Disque Dur
 temporaire

> Les variables :

• Déclaration :

Type identificateur; ex: int a; int a=1;
 [0,255] initialiser.

• Types :

int %d decimal.
 long %ld
 unsigned char %u
 Double %lf virgule flottante
 float %f
 char %c

> Les constantes :

• Déclaration :

const Type identificateur = valeur

> Les calculs de base :

+ - * (x) / (÷) % modulo

= pour l'affectation.

• autres opérations :

a++ : incrémentation.

a-- : Décrémentation.

op = valeur
 opérand

ex: a = a x 2
 a = a + 100
 Simply
 a x = 2
 a + = 100

// ou /*...*/ : comment.

\n : retour à la ligne

\t : tabulation

"..." : afficher.

␣ : () à la fin de chaque instruction.

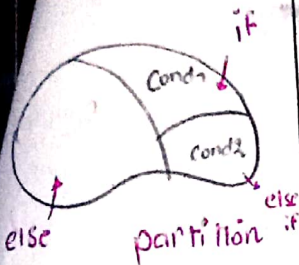
␣ : le C fait la diff entre 'A' et 'a'.

printf : afficher le texte

scanf : récupérer une saisie

formaté

> Les conditions:



@: on met pas

(;) à la fin de la condition

~~if (age > 70);~~

> Variable globale:

• accessible à tous les fichiers:

on la met au haut du fichier après les includes

• accessible à le fichier uniquement:

on ajoute le mot clé **Static**.

mais @ elle conserve toujours la valeur de la dernière fois...

> Diff entre "while"

"w.do":

while: elle peut ne s'exécuter pas
do while: elle s'exécute au moins une fois.

• il y a pas la notion du procédure en langage C, mais on peut la réaliser en utilisant les pointeurs.

• @: n'oublions pas de retourner le résultat à la fin de votre fonction.

```
• if ( cond1 )
{
    cond1 ✓
}
else if ( cond2 )
{
    cond2 ✓
}
else
{
    cond1 & cond2 X
}
```

menu

Switch:

Switch (m)

Case V1:

-----;

break;

Case V2:

-----;

break;

default:

-----;

break;

afin de ne pas répéter tout le block "n".
on utilise la notion de switch, dans la quelle on teste la valeur d'une seule variable.

• Condition ternaire:

if (majeur) ^{Boolean}

majeur = 1 age = 18

else
majeur = 0 age = 17

ma condition
age = (majeur)? 18 : 17;
ma variable
else

> Les boucles:

while:

Tant que

for:

Pour

while .. do:

do that while ed is true

> Les fonctions:

• La structure d'une fonction:

```
Type identif (para)
{
    // instructions
}
```

ma fonction renvoie

Line
Valeur

rien

void

Terminé.

Static fonction

→ cette fon est locale

> Notons que les cellules de la RAM sont numérotées de 0 à (taille) un très grand nombre!

> op: autorise l'utilisation de la mémoire..

* à chaque fois qu'on veut utiliser notre variable, elle sera remplacée par son ADR intervient le

code de compilateur

> @: il est préférable d'initialiser ses pointeur avec NULL

La case est vide.

> @:

*p ⇒ le contenu
p ⇒ l'adresse
&x.

@:

char lettre = 'A'
printf("%c", lettre)

code 65
ASCII

printf("%c", lettre)

A.

> Donc pour écrire un mot de 5 caractères il faut un tableau de 6 cases.

String.h

La bibliothèque qui contient les fonctions sur les chaînes

> Les pointeurs:

• pour connaître l'adresse:

ADR	Contenu
0	--
1	--
...	--
50	50

X

RAM.

→ @ : à chaque case on peut stocker un & un seul mot.

cloud: \$x → %p : en hexadécimal.
→ %d : en décimal.

cloud: un pointeur: une variable qui contient une adresse.

• Déclaration d'un pointeur:

Type * identificateur; ou Type * identificateur;

ex: int * p1, p2; ou int * p1, * p2;

> Il est préférable que le type de votre variable & votre pointeur soit le même

de la var bien sûr.

> Deux pointeurs qui contiennent la même adresse, pointent évidemment sur la même variable

• en voyer un pointeur vers une fonction:

modification direct du contenu de la variable.

ex: int main(*m)
{ int m = 5;

cette fonction reçoit un pointeur
Triplet (m); l'adresse de m.
"une adresse" & printf("%d", m);
modifie la valeur
return 0;

> Les chaînes de caractères:

Une chaîne de caractère n'est rien d'autre qu'un vecteur de type char → Ce vecteur se termine par '\0'.

• quelques fonctions:

strlen: longueur.

strcpy: copier une chaîne dans une autre.

strstr: rechercher une chaîne dans une autre.

printf("%s", chaîne)
scanf("%s", chaîne).

indices commencent par 0.

Tableau de 3 cases

RAM

> par défaut, chaque case est initialisée à 0.

programme qui s'exécute juste après la compilation.

elle prend rien de la mémoire, contrairement à les constantes... on peut l'utiliser pour déterminer la taille d'un tab.

Constantes prédéfinies

- LINE - : numéro de ligne actuelle.
- FILE - : nom du fichier actuelle
- DATE - : date de compilation.
- TIME - : heure de compilation

Directement

> Les tableaux :

- Créer un Tableau :

type identificateur [taille]

- accéder à une case :

Tableau [Numero de la case]

- initialiser un Tableau :

ex: int tableau [4] = { 0, 7, 2, 3 } ;

```
printf("%d", tableau)
    ↓
l'adresse de la première case.
printf("%d", tableau)
    ↓
sa valeur
*(tableau + 1)
    ↓
la valeur de la deuxième case.
```

> Les préprocesseurs :

- # Define :

elle permet d'ajouter une valeur à un mot, elle prend rien de la mémoire.

ex: # Define Max 100.

→ on peut aussi faire des calculs sur les Define.

- Les macros : (en utilisant les Define).

on peut remplacer un mot par un code source.

ex: # Define coucou() printf(...);
printf(...);

→ avant la nouvelle line.

- Les conditions : on peut faire des

compilations conditionnelles.

```
#if cond1 /* code source à exécuter */
# elif cond2 /* code source à exécuter */
#endif
```

une boucle infinie { A C B }
{ B C A }

- ifdef, ifndef :


#ifdef window → code ?
#endif.

#ifndef

pour éviter les inclusions infinies.

une façon
de déclarer une
variable.

char occupe un ^{ex}
octet.
int occupe 4 octets

case
mémoire = 1 octet
in fact, 

> L'allocation Dynamique: réserver une place nécessaire
à la mémoire (même si on connaît pas combien) → taille de
tableau.

• **sizeof (...)** : Savoir le ^{mon} type d'un type.

• Allocation de mémoire dynamique: →

Demander de
la mémoire
manuellement

malloc: demander de l'os la permission d'utiliser
un peu de la mémoire.
free: dire à l'os qu'on a plus besoin de cette
mémoire.

Demarche: malloc → vérifier si malloc a
réussi → free.
malloc renvoie une adresse → NULL
on l'envoie l'adresse mémoire à libérer

> Les Limites de scanf:

ex: scanf ("%s", &nom) Si je tape
printf ("%s", nom) DANIA Adimi
elle affiche que

⚠: la fonction scanf s'arrête si elle tombe au cours
de sa lecture sur un espace.

C'est pourquoi on a:

gets:

elle écrit toute
la chaîne mais
elle contrôle pas
les buffer
overflow.

fgets:

gets, mais en
version plus
sécurisée.

Tableau
de 10 cases.
fgets (nom, 10, stdin)
taille MAX input
Dania Adimi

printf ("...", nom).

+ elle vas
écrire dans des
zones mémoires
non prévues.

buffer overflow.

> Comment créer ses propres bibs:

> prototype: une indication pour l'ordinateur pour s'organiser.

int airRectangle (long a, long b);

> Les fichiers:

Les headers

.h:

contiennent les
prototypes des
fonctions.

↓
chaque
fichiers .c
ont ses équivalents
.h

.c:

contiennent les fonctions
elles mêmes.

on la
compile puis
on la sauvegarde
.h

```
#ifndef NOM-DU-BiB
#define NOM-DU-BiB
#include <stdlib.h>
#include <stdio.h>
..... Les prototypes des fonctions .....
.....
..... Les corps des fonctions .....
.....
#endif
```

mais:

elle doit
se trouver dans
le repertoire
où se trouve
mon
programme.

➡ Dans mon programme principal il
suffit de faire #include "NOM-DU-BiB"

Structure.

on la met dans

(,)

pour un jeu

2D

- le nom de l'équivalent

comme si un nouveau type!

char x;

nom de mon équivalent

⚠ : il est préférable d'initialiser notre structure.

ex: coordonnees point = {0,0}

> Créer votre propre type de variables:

• Structure: un assemblage des variables de type différent

• Déclaration d'une structure:

struct

{

}

;

mes variables avec leurs types!

ex:

struct coordonnees

{

int x;

int y;

};

• L'utilisation d'une structure:

typedef → on le met juste avant la définition de notre structure.

typedef struct

son nom

{

}

;

le nom de

l'équivalent.

coordonnees A;

A.x

A.y

• modifier les composants d'une structure:

variable • nom de composant

ex: point.x = 10

• envoyer une structure à une fonction: qui vas modifier..

(*point).x = 0

Score d'un jeu par ex.

> Les fichiers: pour stocker nos informations même si le programme est arrêté.