Chaque processeur a 2 utilisation (effectif et réel).

Flemme d'écrire l'architecture arm. (a2:50)

CM2:

Codage int dépend de la machine pas spécifié par le langage C (on peut quand même le spécifier).

Utiliser BCB pour op ne devant pas souffrir d'erreur (finance, phy--) représentation exacte des nombres humains / décimaux en base.

en virgule flottante : après virgule $2^{-1}, 2^{-2}$ --- avant $2^0, 2^1$
< 0 ≥ 0

D'abord codé partie entière puis partie décimal

---/----.

Important norme / standardisation car même si codage était pareille les machine en fonction de leur indicateur ne code pas octet dans le même ordre (big indien et low indien processeur code les "7" octets d'un mot de 32 bit différent octet poids fort début ou fin resp).

Donc sur le réseau une norme a été choisi big ou low j'sp.

Plus il y a de bit dans mantisse plus c'est précis 23 bit => 7 chiffres décimaux significatifs.

Si besoin plus passer en double et pas en float.

printf utilise un algo permettant de refaire un arrondi et réafficher nb comme si ils étaient bien codé (pour s'en rendre compte faire des op qui grandira le gap jusqu'à erreur (visible).

Pour voir vrai codage %p (pointeur ainsi on voit vrai représentation).

Par défaut le compil considère littéraux flottant en double (et non float) si on met le littéral dans un float après il fera un cast du double en flot sinon rien.

UTF-8 : moins lourd mais taille variable donc plus complexité (1 à 4 octets).
Universel et lié au UNICode

ASCII lié à l'UTF 8 (mais restreint)

quand le bit fanion d'un octet en UTF8 est un 0 ⇒ auto suffisant octet. octet qui code un symbole (ASCII)

Si c'est un 1 alors ça veut dire plusieurs octet, le nb de 1 informe sur le nombre d'octet constant.

Si ça commence par 10 ⇒ c'est un octet suivant. Permet de recevoir des messages avec des erreurs sans perdre tt les caractères

ex erreur symbole 1

1100 1010 ---- ---- (lecture recommence au prochain octet ne commençant pas par 10)

~~juste 1 symbole perdu~~ reste bon

Les caractères nouveaux sont codés avec le plus de bit.

Structuration type: (35:00)

~~2 grand~~

contiguité : facile accès grâce à calcul simple | adresse tab + sizeof(elem) × indice

taille variable plus compliqué :
↳ chainage o

avantage : ≠ type + allocation dynamique valeur
(≠ structure)

Apnté :
vector + complexe que primitif mais très bien opti
toutefois certain op seront en O(n) car on devra resserrer tt le vecteur donc en th mais ref

en C mise à part le char la taille n'est pas toujours normé

sizeof(char) = 1 octet = 1 byte ; si on lit un char et que le caractère est codé sur plus d'un octet (spéciaux) on ne verra que le premier octet du caractère

Tab de char (en C) terminé par '\0' comme pas de classe on a pas attendre rien donc on doit faire appel à fonction tierce qui compte ....

char *s = "é" ; sizeof = 2 octet pour char spécial + 1 : '\0'
= 3

sizeof n'est pas une fonction c'est un opérateur static (on connait au moment compil).

soit taille_tab soit délimiteur ex : env est un char ** et le main a pas taille de env donc env[taille-1] = NULL ;

'\0' != NULL
1 octet à 0 sizeof(ptr) à 0
00000000
Attention : conversion implicite
warning all pour être en premunir

struct et Union :
Syntaxiquement égaux
Sémantiquement !=

struct anonyme struct { ... } x ; | déclaration de x une var.
x est de type "la struct qui est au dessus

- 53:00

Union = "soit j'ai ça soit j'ai ça" XOR d'un des membres de l'union

ex :

```
union { int âge
        char* nom
        bool sexe };
```

Union pour récolter info d'un fichier JSON qui représente une bdd avec les attributs nom, âge, sexe.
Pas aller à std::variant dans ce cas

1 champ implémenté pas les 3
!= struct concat champ

définition : ⇒ réserver espace mem pour une var

int i ; // = déf non initialisante

que pour global logique

extern int i ; // déclaration, elle est connue mais dans une autre unité compilation

pour les fonctions + facile :

prototype : déclaration

proto + corps : définition car on peut déterminer la taille pour la stocker

les var globales sont dans le "segment statique" ⇒ initialisé par défaut (à 0).

si dans une fonction une var n'est pas initialisé elles peuvent prendre la val des objets fantômes (vue 1ère CM).

Compilation :

Chaque source c va donnée un fichier objet (fo : fichier binaire donc non lisible par être humain)

Tout ces fichiers binaire (.o) sont liés entre eux par fabriqué le "main".

Les options de l'éditeur de lien comme "-lm" (pour la lib math) doivent être en fin de ligne de compilation car -lm en amont serons passé à ld (éditeur de lien) par gcc.

On peut le faire à la main (cf CM 1) une des raisons pourquoi on laisse gcc s'en occuper pour inclure H les lib standard (via option passé à ld comme "-lm").

Prétraitement et Preprocessing, uniquement des substitutions de texte (rien de intéressant) : inclusion fichier en tête, macro (#define) - --

Eviter définition fichier en tête (car double def ⇒ erreur ex :

" include bad-header dans toto.c
include bad-header dans toto.c } edition lien ⇒ 2 def fonction "

Meynard 2020 je crois que c'est faux, sinon à quoi sert ifndef devant header

Rappel :

argc : nombre d'éléments de la ligne de commande

argv : contient ces éléments sous la forme d'un tableau de chaine de caractères
(argv[0] : nom de la commande et pour i allant de 1 à argc - 1 le ième arg de la commande)

argc/env : tableau de chaine de char stockant l'environnement (contexte commande)

Code assembleur :

Une code instruction (ex : pushl, movl, ...) → pousse dans la pile

pushl %ebp ; pousse dans la pile ebp ebp : registre de base de pile dans ptr vers bloc courant de pile.

Rappel : bloc pour chaque ptr fonction

subl $8 reserve 8 octet pour var local (en l'occurence n)

(%ebp) : % l'endroit pointé par registre 8(%ebp) : % avec decalage de 8