IAP TD 1

Exercice 1

$2 \times 5 + 20 \,\%\, 7 + 13 - 12$

$10 + 7 + 13 - 12$

$-2 + 13 - 12$

$11 - 12$

0

Exercice 2

$x = 1;$

$y = (1 + 1) + (5 \times 3)$

$z = (a + a) \times 2$

$y = (1 + 1) + (5 \times 3)$

$= 2 + 15$

$= 17$

$z = (2 + 1) \times 2$

$= 2 \times 2$

$= 4$

$a = 2 \quad b = 16$

L1 : x = 1 ; y = ? ; z = ?

L2 : x = 1 ; y = 16 ; z = ?

L3 : x = 2 ; y = 16 ; z = 6

$(a == 0) \;\&\&\; (a < b) \;\&\&\; a >= 9 \;||\; (a > b \;\&\&\; a == 0)$

F F F

a → nb année

(a%4==0) && (!(a%100==0)
|| (a%400==0) && (a%4000==0)

Exercice 4

{ printf("Donner
int d = 100;
int e = 0;
float t = 1,0965;
e = 100 * t;
printf("100 dollars = %f euros" e);
}

{
int d = 100, e = 0;
float t = 1,0965;
e = 100/t;
printf("%d dollars = %f euros", d, e);
}

{double d = 100, eu = 0;

Exercice 5

int a, m;
printf("Donner l'année : ");
scanf("%d", &a)

le mois

{
int a, m, nbj = 0;
printf("Donner le mois et l'année");
scanf("%d %d", &m, &a)
if (a%4==0) && (!(a%400==0) || (a%4000==0)) &&
m = 2;
nbj = 29
else
if m==4 || m==6 || m==9 || m==11;
nbj = 30
else nbj = 31

ALGORITHMIQUE IAP 7

- Architecture de Von Neumann
- Source ("fichier".txt, .c) → langage haut niveau → entrainé des machines en mémoire. Machine de Turing. Traduction en langage machine pour l'exécution → compilation → édition de lien → fichier .exe
  OU
  Interpréteur (traducteur) → ex : python. langage interprété
- Compilation
- Étape de la compilation :
  - Analyse lexicale : mots du langage
  - Analyse syntaxique : syntaxe
  - Analyse sémantique : Structure de donnée
  - Transfo code intermédiaire
- Édition de lien :
  Construire image mémoire des sources logiciel compilés séparément
- Langage :
  - Mots
  - Syntaxe
  - Sémantique (sens)

  ↳ Spécifier dans un manuel de référence.
- BNF
  - ::= def cat syntax
  - <> délimite d'une cat
  - | ou logique
  - {} répétition 0 ou plusieurs fois
  - [] exception 0 ou 1 fois

Diagrammes de Contrôle

parcours fléché, aller d'un point à l'autre

Test de logiciel

Très compliqué mais crucial

Génie logiciel

→ Aboutir à un code fiable et opti

- spécification ⟶ Validation + plans de validation
- conception ⟶ intégration + plans d'intégration
- codage et test unitaire

TD2

TP

Exercice 1

Saisir n
i = 1
fact = 1
tant que i ⩽ n
    fact = fact × i
    ++i
Afficher fact

```
int main() {
unsigned int n, i=1, fact=1;
scanf("%u", &n);
while (i <= n) {
    fact *= i;
    ++i;
}
printf("%u! = %u", n, fact);
}
```

---

saisir n
fact = 0
pour i allant de 1 à n-1
    fact = fact × i
Afficher fact

```
int main() {
unsigned int n, f, i;
scanf("%u", &n);
for (f=1, i=1; i<=n; i++) {
    f *= i;
}
printf("%u! = %u", n, f);
}
```

Exercice 2

Saisir n
i = 1
tant que i ⩾ n
    si n%i == 0
        afficher i
    i++

```
int main() {
unsigned int n, i;
i = 1;
while i <= n {
    if (n % i == 0) {
        printf("%u ", i);
    }
    i++
}
printf("\n");
}
```

Lire (a, b)
Si b == 0
Afficher PGCD = (a, 0)
Sinon
Afficher PGCD (b, a % b)

```
int main() {
scanf(
if b == 0 {
printf "PGCD
```

```
int main() {
unsigned int a, b;
scanf("%u %u", &a, &b);
if b == 0 {
printf("PGCD = (%u, 0)", a);
else {
printf("PGCD = (%u, %u)", b, a%b);
}
```

IAP TD 3

Exercice 1

1.1) Personne* personnes[MAX_PERSONNES]

1.2) initialiser

```
/**
 *@brief initialiser le tableau à vide
 *@param [in-out] tableau
 *@param [in] MAX_P
 */
void initialiser(Personne** table, unsigned int taille
    unsigned int i;
    for(i=0; i<taille; ++i){
        table[i] = NULL;
    }
```

1.3) /**
 *brief ajoute une donnée à l'indice i
 *

personnes[i] = new Personne;
cout << "Saisir (MAX_NOM) de la personne n°" << i << endl;
cin >> saisie(MAX_NOM) >> buffer;
// Allocation dynamique
for(unsigned j=0; buffer[j] != '\0'; ++j){
    pers->nom[j] = buffer[j];
}
cout << "Entrez l'age de la personne";
cin >> pers->age;

```
personnes[i] = new Personne;
cout << "Entrez le nom :" << endl;
cin >> setw(MAX_NOM) >> buffer;
personnes[i]->nom = new char[strlen(buffer)+1];
strcpy(personnes[i]->nom, buffer);
cout << "Entrez l'age :" << endl;
cin >> age;
```

IAP

TD 3

1.1)

```
#include "stdlib.h"
#include "stdio.h"
#include "math.h"

int main(){

    double a=3, b=5., c;

    c=pow(a,b);
    printf("%f^%f = %f\n", a, b, c);

    system("pause");
    return 0;

}
```

1.2)

```
#include "stdio.h"
#include "stdlib.h"
unsigned int factorielle(unsigned int n){
unsigned int i, fact;
for(i=0, fact=1; i<n; i++){
    fact *= i+1;
}
return fact
}
```

```
int main() {
printf("01 = %d, 41 = %d, 51 = %d",
  factorielle(0), factorielle(4), factorielle(5));
```

Exercice 2 :

L'affichage est x = 1 car les variables se vident après la fin de l'exécution de la fonction.

Exercice 3 :

			Affichage
L11	x [3]	y [5]	
L12	x [3]	y [5]	3,5
L13	x [3]	y [5]	5,3
L14	x [3]	y [5]	3,5

Elle ne fonctionne pas comme prévu.

Exercice 4

```
int division(int a, b) {
unsigned int res;
while a>b {
    a = a-b;
    res++;
}
return res;

int main() {
  int x, y;
  scanf("%d %d", &x, &y);
  printf("%d/%d = %d \n", x, y, division(x,
```

Exercice 5

```
unsigned int h(n) {
  if (n%2==0) {
    return n/2;
  }
  else {
    return 3n+1
  }
unsigned int g(n) {
  if (n==1) {
    return 0;
  }
  else {
    return 1+g(h(n));
```

Prototypage de fonction

1) Quel nom?
2) De quelle données a-t on besoin pour résoudre le pb?

Entrées [in]: - utilisés pour le traitement
- caractéristique: dans le corps de la fonction en partie droite d'affectation et/ou condition

3) Quelles sont les ~~conditions~~ modes de transmission des résultats de la fonction

a) transmis par le retour [return] unique résultat envoyé par return
b) transmis par le paramètre de sortie [out]: Mode de passage par pointeurs utilisés dans le corps de la fonction en partie gauche d'affectation
c) transmis par paramètre entrée/sortie [in-out]

4) Retour de la fonction avec: type, non: void
5) Ecrire le nom de la fonction
6) Écrire les deux paramètres formels entre parenthèses

[in] [out] [in out]

↳ type/nom du paramètre
↳ type* → nom du paramètre

```
Précondition : n>0, taille>0, n≤taille

int nieme(int* t, unsigned int n, unsigned int taille)
{
    assert( n>0 && taille>0 && n≤taille);
    return t[n-1];
}

int plusgrand(int *t, unsigned int taille)
    assert(taille>0);
    unsigned int i;
    int temp=t[0];
    for (i=0; i≤taille; i++) {
        if temp<t[i] {
            temp= t[i];
                    }
                        }
    return temp;

int indice(int x, int* t, unsigned int
taille);
assert(
int i=0
while(x != t[i] && i<taille){
    i++;
                    }
if (i==taille ){
    return -1;
                    }
else {
    return i;
    }
```

```
bool minore(const int *t, unsigned int taille, int a){
  assert(taille > 0); int minimum;
  for(unsigned i=0; i<taille; ++i){
      if(t[i] < minimum
          minimum = t[i];                    }
  if(a == minimum){
    return TRUE; }
  else{
    return FALSE;
      }
                                             }
```

c'est vrai ?

```
bool dateValide(const Date* d){
  if( d->jour<31 && d->mois<12 &&
```

TAP — TD 4

Exercice 1 :

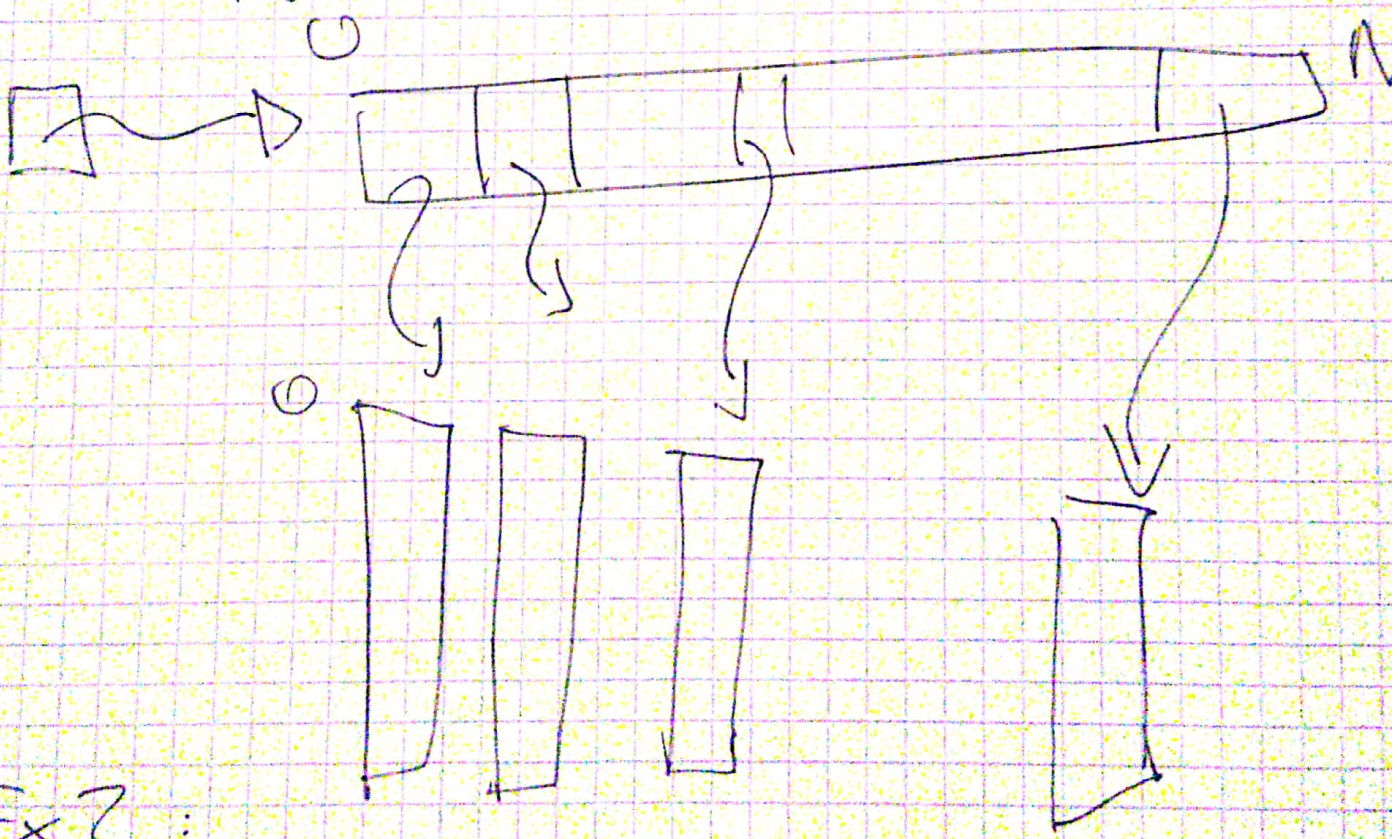
SontAmis(entier a, entier b)
→ retourne booléen

DevenirAmi(entier a, entier b)

Entier N = 7.10^9 // nb habitants

Taille du booléen de taille N-1 : tab1
// tableau de relat° d'1 pers avec tt le monde

Tableau entier du tableau de (X) de taille N-1.



Ex 2 :

amis[ ] {0, 1, 2, 3}

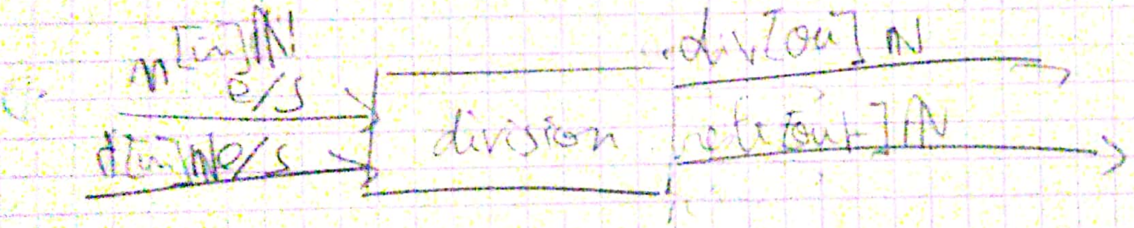
Initialisation (tableau d'Entier T de taille N):
Entier i
Faire i allant de i ← 0 à N-1:
    T[i] ← i

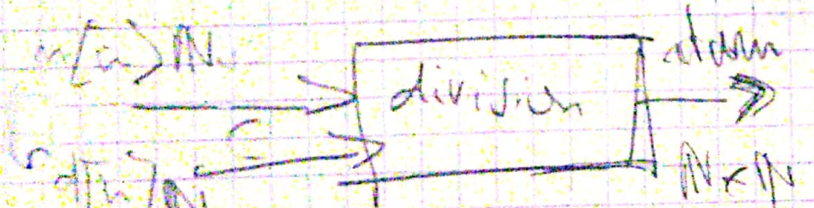
Fonction rep2 (Entier rep)
    -> Retourne Entier
Entier Rep
Tant que (T[Rep] == rep)
    Rep ← T[Rep]
Retourne Rep

IAP TD5

Exercice 4

```
void division(unsigned int*a, unsigned int*b){
  unsigned int temp;
  *temp=*a/*b;
  *b=*a%*b;
  *a=*temp
}
void division(unsigned int a, unsigned int b){
  printf("%u / %u Q=%u R=%u", a, b, a/b, a%b);
}
```





avec un struct qui contient quotient et reste

& dir ou est rangé une valeur

et sinon on mix des 2