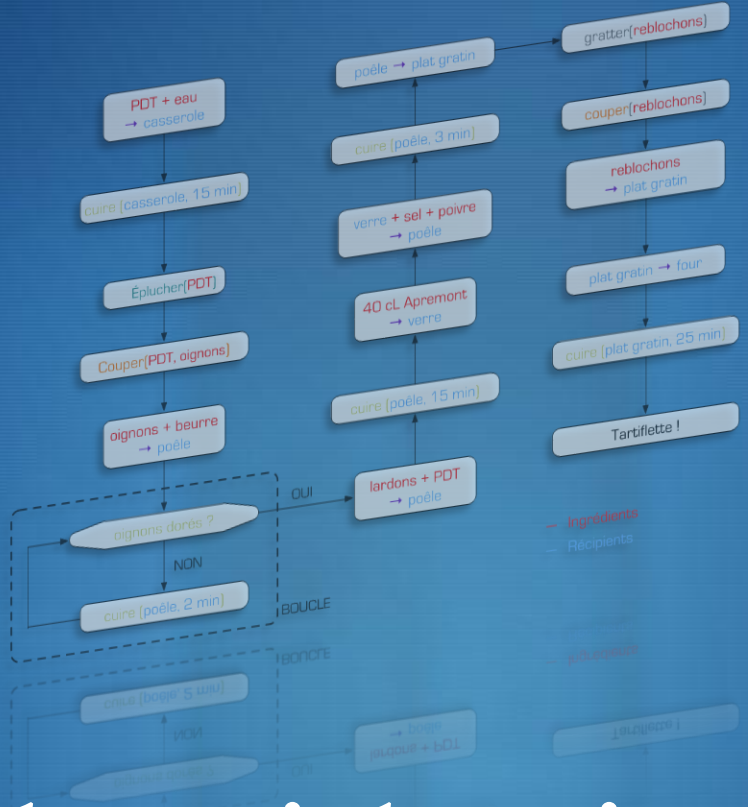


Algorithmique

2. Variables et affectation

G. GEVRIN

SOMMAIRE

- Pseudo code
- Codage binaire
- Logique booléenne
- Opérations arithmétiques

Pseudo code

- Intérêts du pseudo-code :
  - Clair, lisible
  - Pédagogique
  - Indépendant du langage de programmation
  - Pour distinguer le fond de la forme

Syntaxe

- Déclaration d'un algorithme :

```
PROGRAMME Addition
Entrées : entiers i et j
Sortie : entier
DEBUT
        ...
        renvoyer ...
FIN
```

Syntaxe

- Déclaration d'une variable :

```
Variables : entier i
```

- Affectation d'une variable :

```
i ← 42
```

Syntaxe

- Test :

```
SI i = 42 ALORS
        …
SINON
        …
FIN SI
```

- Boucle :

```
TANT QUE i > 3 FAIRE
        …
FIN TANT QUE
```

Puissance de 2

- Résultat attendu :

```
DeuxPuissance(4) = 16
```

- Intuition :
  - Tant qu'on n'est pas à la puissance voulue, on …
  - Au total, pour DeuxPuissance(*a*), on fait … multiplications par 2.

Puissance de 2

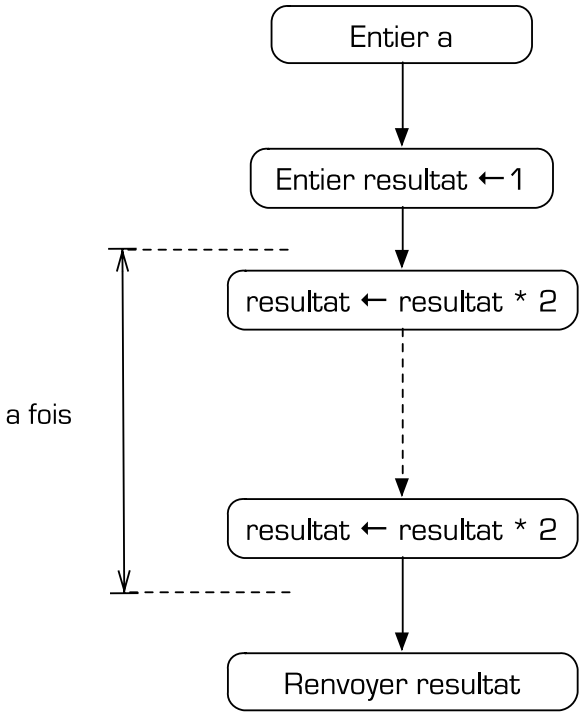
- Résultat attendu :

```
DeuxPuissance(4) = 2*2*2*2 = 16
```

- Intuition :
  - Tant qu'on n'est pas à la puissance voulue, on multiplie par 2
  - Au total, pour DeuxPuissance(*a*), on fait *a* multiplications par 2.

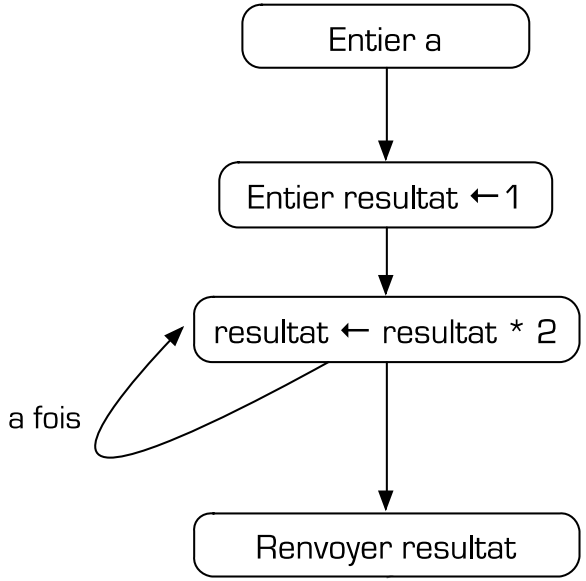
Puissance de 2

- En français : je multiplie *a* fois par 2 l'entier1
- Organigramme :



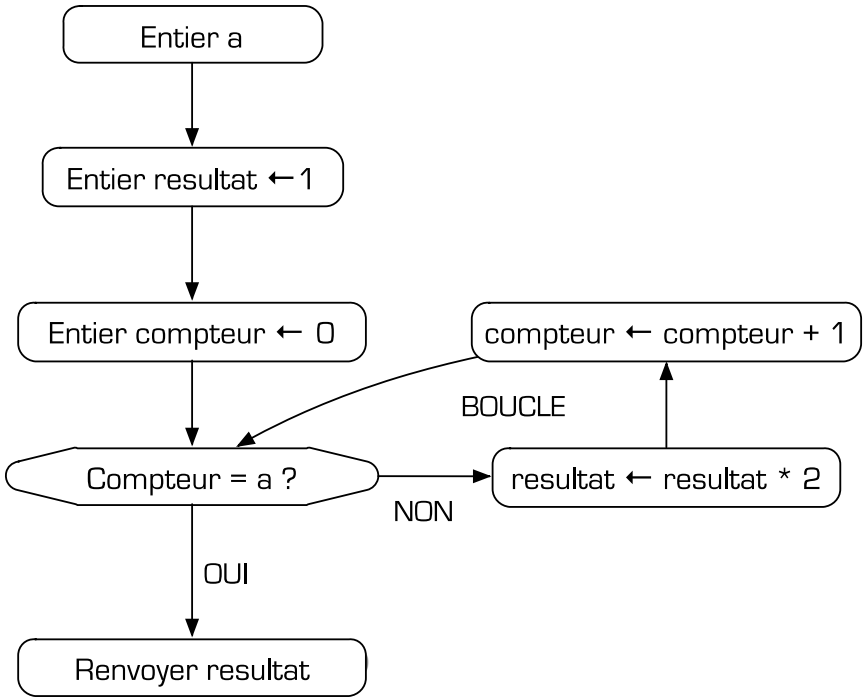
Puissance de 2

- En français : je multiplie *a* fois par 2 l'entier1
- Organigramme :



Puissance de 2

- En français : je multiplie *a* fois par 2 l'entier1
- Organigramme :



Puissance de 2

- En français : je multiplie *a* fois par 2 l'entier1

- Algorithme :

```
PROGRAMME DeuxPuissance
ENTREE : entier a
SORTIE : entier
VARIABLES :
        entier compteur, resultat
DEBUT
        compteur ← 0
        resultat ← 1
        TANT QUE compteur < a FAIRE
                resultat ← 2 * resultat
                compteur ← compteur + 1
        FIN TANT QUE
        RETOURNE resultat
FIN
```

Puissance de 2

- Programme en C :

```c
int DeuxPuissance(int a)
{
    int compteur = 0 ;
    int resultat = 1 ;

    while (compteur < a)
    {
        resultat = resultat * 2 ;
        compteur = compteur + 1 ;
    }
    return resultat;
}
```

Exemple tordu

- Algorithme :

```
PROGRAMME Tordu
VARIABLES :
        entier i
DEBUT
        i ← 1
        TANT QUE i > 0 FAIRE
                i ← 2 * i
                AFFICHE(i)
        FIN TANT QUE
        AFFICHE("J'ai fini !")
FIN
```

Codage

- Ce programme affiche-t-il "J'ai fini !" ?

```c
int main(int argc, const char * argv[]) {
    int i = 1 ;

    while ( i > 0 )
    {
        i = i * 2 ;
        printf("%d\n", i);
    }

    printf("J'ai fini !") ;

}
```

Codage

- Ce programme affiche-t-il "J'ai fini !" ?

```c
int main(int argc, const char * argv[]) {
    int i = 1 ;

    while ( i > 0 )
    {
        i = i * 2 ;
        printf("%d\n", i);
    }

    printf("J'ai fini !") ;

}
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
J'ai fini !Program ended with exit code: 0
```

Codage binaire

- Variables de type entier ne sont pas de "vrais entiers" mais appartiennent à un intervalle :
  - [-2147483647 , 2147483647] ou [$-(2^{31} - 1)$, $2^{31} - 1$]

Codage binaire

- Toutes les données traitées sont codées en binaire : 0 ou 1
- Exemple : 1101100001101
- Ecriture binaire (base 2) → Ecriture décimale (base 10) ??

Codage binaire

- Exemple : 1101100001101

0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

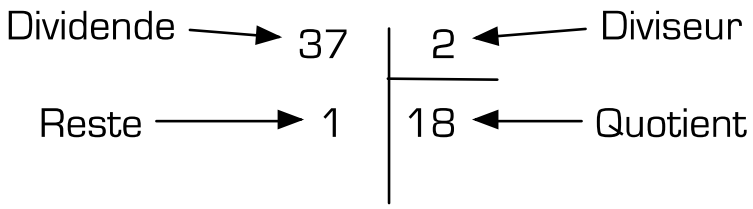
- $2^{12}+2^{11}+2^{9}+2^{8}+2^{3}+2^{2}+2^{0}=4096+2048+512+256+8+4+1$
- $2^{12}+2^{11}+2^{9}+2^{8}+2^{3}+2^{2}+2^{0}=6925$

Codage binaire

- Ecriture décimale (base 10) → Ecriture binaire(base 2) ??
- Convertir 37 en binaire

Codage binaire

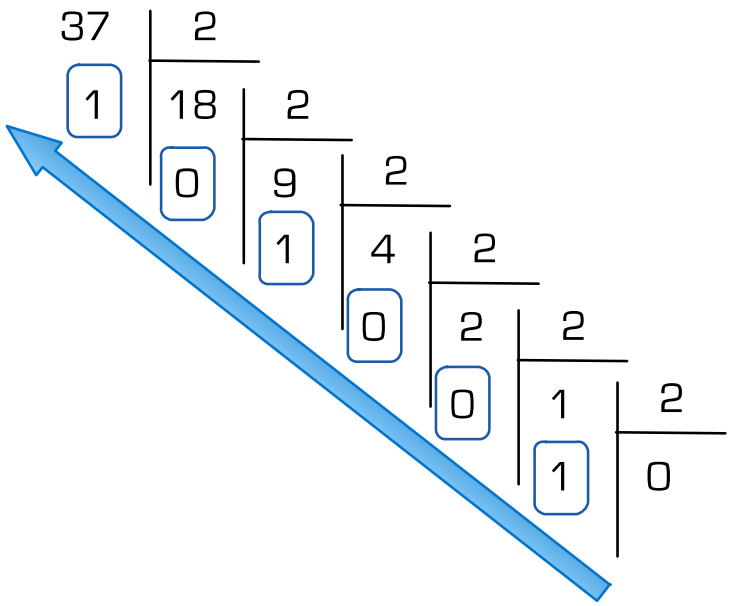
- Ecriture décimale (base 10) → Ecriture binaire(base 2) ??
- Division Euclidienne :



  - Reste en C : 37%2
  - Quotient en C : 37/2

Codage binaire

- Ecriture décimale (base 10) → Ecriture binaire(base 2) ??
- Division Euclidienne :



37 (décimal) → 100101 (binaire)

Codage binaire

- 1 bit = 0 ou 1
- 1 octet (byte) = 8 bits
- 1 Ko = 1024 octets
- 1 Mo = 1024 Ko
- 1 Go = 1024 Mo
- 1 To = 1024 Go
- 1 Po = 1024 To
- 1 Eo (Exaoctet) = 1024 Po
- 1 Zo (Zettaoctet) = 1024 Eo
- 1 Yo (Yottaoctet) = 1024 Zo

Caractères

- Caractère : 'a', 'A', '7'
- Chaîne de caractères :
  - "On connaît, dans les grandes cours, un autre moyen de se grandir: c'est de se courber."
  - "a"
  - "42"
- Implémentation : ça dépend...
- Opérations : concaténation

Logique booléenne

- George BOOLE (1815-1864)
- B={1 , 0}
- Opérations :
  - ET (AND)
  - OU (OR)
  - NON (NOT)
  - OU EXCLUSIF (XOR)

Logique booléenne

- Tables de vérité :

ET	VRAI	FAUX
VRAI	VRAI	FAUX
FAUX	FAUX	FAUX

OU	VRAI	FAUX
VRAI	VRAI	VRAI
FAUX	VRAI	FAUX

XOR	VRAI	FAUX
VRAI	FAUX	VRAI
FAUX	VRAI	FAUX

NON	VRAI	FAUX
	FAUX	VRAI

Opérations arithmétiques

- Type entier : + , - , * , / (division entière) , % (modulo), ^ (puissance)
- Type flottant : + , - , * , /
- Type booléen : && (AND), || (OR), ! (NOT)

Exercice : terminaison

- Problème : aller en voiture d'ici à Part-Dieu
- Indice : à tout moment, on peut voir la tour
- Proposition : prendre toujours la rue qui se dirige vers la bonne direction

→ Cet algorithme se termine-t-il ?

Exercice : terminaison

- Proposition : prendre toujours la rue qui se dirige vers la bonne direction : algorithme glouton (greedy)
- Algo gloutons : faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global. Ne fournit pas systématiquement la solution optimale

→ Cet algorithme se termine-t-il ? Pas forcément…