

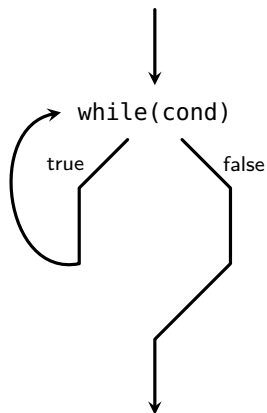
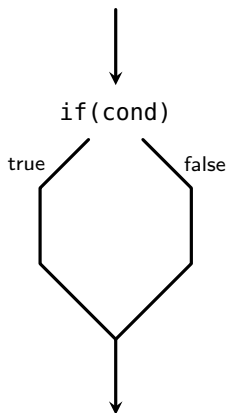
Algo/Prog – DUT 1

- 1 Introduction
- 2 Premiers programmes
- 1 Définition de procédures et fonctions
- 2 Conditionnelle (vu en TP)
- 1 Instructions complexe (boucles)
- 6 Variables, types
- 7 Représentation des données en mémoire
- 8 Boucles imbriquées et tableaux à doubles entrées
- 9 Pointeurs, etc

La boucle while

- jusqu'à maintenant : séquence d'instruction avec aiguillage (`if`)
- comment répéter (*itérer*) un traitement plusieurs fois ?
- nombre d'itérations variant d'une exécution à l'autre ?
- notion de répétition d'une séquence d'instruction ?
- quand arrêter les itérations ?
 - quand le calcul répétitif est fini
 - = lorsqu'une condition devient fausse.

Flot des instructions



La boucle while

- Syntaxe comparable à un `if` sans `else` :

```
while (cond) {  
    ... // Corps de la boucle  
}
```

La boucle while

- Syntaxe comparable à un `if` sans `else` :

```
while (cond) {  
    ... // Corps de la boucle  
}
```

Surtout pas de point virgule ici!

Flot des instructions

```
while(cond){  
  
    printf(...);  
    x = ...;  
    ...  
    ...  
  
}
```

Flot des instructions



```
while(cond){
```

```
    printf(...);
```

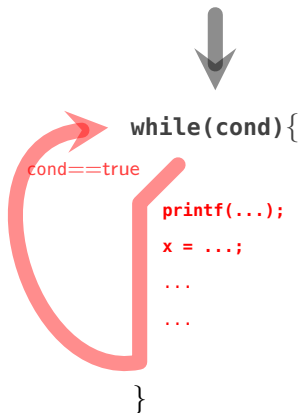
```
    x = ...;
```

```
    ...
```

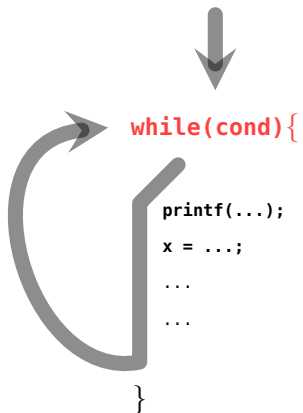
```
    ...
```

```
}
```

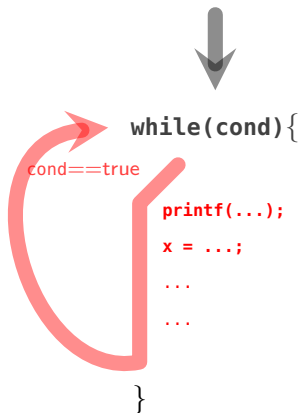

Flot des instructions



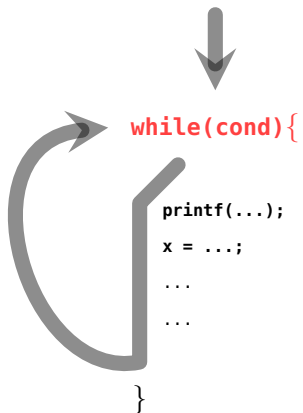
Flot des instructions



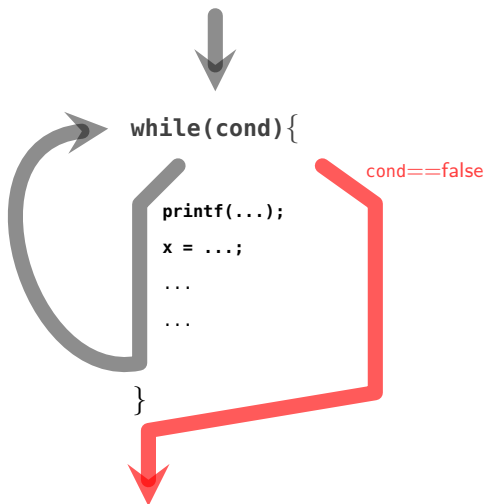
Flot des instructions



Flot des instructions



Flot des instructions



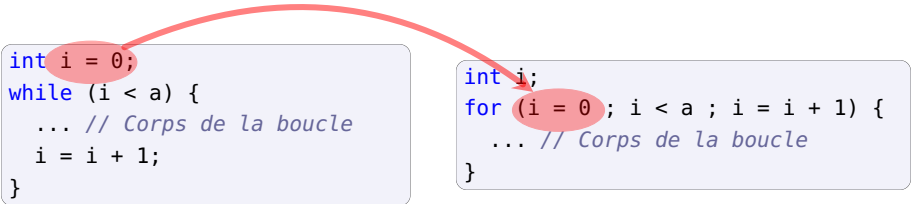
La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```

```
int i;
for (i = 0 ; i < a ; i = i + 1) {
    ... // Corps de la boucle
}
```

La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```



```
int i;
for (i = 0 ; i < a ; i = i + 1) {
    ... // Corps de la boucle
}
```

La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```

```
int i;
for (i = 0; i < a; i = i + 1) {
    ... // Corps de la boucle
}
```


La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```

```
int i;
for (i = 0 ; i < a ; i = i + 1) {
    ... // Corps de la boucle
}
```

La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```

```
int i;
for (i = 0 ; i < a ; i = i + 1) {
    ... // Corps de la boucle
}
```

instructions!

expression!

La boucle for – un cas particulier du while

```
int i = 0;
while (i < a) {
    ... // Corps de la boucle
    i = i + 1;
}
```

```
int i;
for (i = 0 ; i < a ; i = i + 1) {
    ... // Corps de la boucle
}
```


- `for` Réservée aux boucles avec manipulations simples d'indices

la condition

- `cond` condition booléenne quelconque
- comme pour un `if`
- `< > <= ==` ... opérateurs de comparaison numérique. ex : `x < 0`
- `&& || !` ... opérateurs booléens ex : `x < 0 || x > 9`

```
while (x < 8 && !(y == f(z)) || u) {  
    ... // Corps de la boucle  
}
```

Les tableaux

- une variable qui contient plusieurs valeurs d'un même type
- si n valeurs, numérotées de 0 à $n - 1$ 
- la i^e case d'un tableau `t`, (notée `t[i]`) = équivalente à une variable.
 - ▶ affectation `t[i] = 12 + f(x);`
 - ▶ utilisation `x = (t[i] - 1) * u[j];`
- Déclaration :

```
int t [9]; // 9 cases  
int t = {9,8,7,6,5,4,3,2,1}; //9 cases
```

Parcours d'un tableau

```
int t [9]; // 9 cases
int t = {9,8,7,6,5,4,3,2,1}; //9 cases
int idx = 0;
while(id<9) {
    t[idx] = 0;
    idx = idx + 1;
}

for(idx=0; idx<9 ; idx = idx + 1){
    t[idx] = idx;
}
```

Parcours d'un tableau

```
int t [9]; // 9 cases
int t = {9,8,7,6,5,4,3,2,1}; //9 cases
int idx = 0;
while(id<9) {
    t[idx] = 0;
    idx = idx + 1;
}

for(idx=0; idx<9 ; idx = idx + 1){
    t[idx] = idx;
}
```

longueur-1 est l'indice de la dernière case

Parcours d'un tableau

```
int t [9]; // 9 cases
int t = {9,8,7,6,5,4,3,2,1}; //9 cases
int idx = 0;
while(id<9) {
    t[idx] = 0;
    idx = idx + 1;
}
```



A horizontal array of 9 cells, each containing the number 0. A red arrow points from the first cell back to the closing brace of the while loop in the code above.

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

```
for(idx=0; idx<9 ; idx = idx + 1){
    t[idx] = idx;
}
```



A horizontal array of 9 cells containing the numbers 0, 1, 2, 3, 4, 5, 6, 7, and 8. A red arrow points from the first cell back to the closing brace of the for loop in the code above.

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

Pattern classiques : « il existe »

Déterminer si il existe une case `i` telle que `test(t[i])` vrai.

```
int idx = 0;
int candidatReponse = FALSE;
while(id<9) {
    if(test[i]) {
        candidatReponse = TRUE;
    }
    idx = idx + 1;
}
```

Pattern classiques : « il existe »

Déterminer si il existe une case `i` telle que `test(t[i])` vrai.

la réponse « pour l'instant »

```
int idx = 0;
int candidatReponse = FALSE;
while(id<9) {
    if(test[i]) {
        candidatReponse = TRUE;
    }
    idx = idx + 1;
}
```

pour l'instant pas de cases OK

Pattern classiques : « il existe »

Déterminer si il existe une case `i` telle que `test(t[i])` vrai.

la réponse « pour l'instant »

```
int idx = 0;
int candidatReponse = FALSE;
while(id<9) {
    if(test[i]) {
        candidatReponse = TRUE;
    }
    idx = idx + 1;
}
```

pour l'instant pas de cases OK

basculement (définitif) du booléen

ne pas remettre à false

Pattern classiques : « il existe »

Déterminer si il existe une case `i` telle que `test(t[i])` vrai.

la réponse « pour l'instant »

```
int idx = 0;
int candidatReponse = FALSE;
while(id<9) {
    if(test[i]) {
        candidatReponse = TRUE;
    }
    idx = idx + 1;
}
```

pour l'instant pas de cases OK

basculement (définitif) du booléen

ne pas remettre à false

candidatReponse = réponse à la question

Pattern classiques : « il existe »

Déterminer si il existe une case `i` telle que `test(t[i])` vrai.

la réponse « pour l'instant »

```
int idx = 0;
int candidatReponse = FALSE;
while(id<9) {
    if(test[i]) {
        candidatReponse = TRUE;
    }
    idx = idx + 1;
}
```

pour l'instant pas de cases OK

basculement (définitif) du booléen

ne pas remettre à false

candidatReponse = réponse à la question

- Remarque : si 0 élément réponse FALSE

Pattern classiques : « pour tout »

déterminer si toutes les cases `i` telles que un `test(t[i])` est vrai.

```
int idx = 0;
int candidatReponse = TRUE;
while(id<9) {
    if(! test[i]) {
        candidatReponse=FALSE;
    }
    idx = idx + 1;
}
```

Pattern classiques : « pour tout »

déterminer si toutes les cases `i` telles que un `test(t[i])` est vrai.

```
int idx = 0;
int candidatReponse = TRUE;
while(id<9) {
    if(! test[i]) {
        candidatReponse=FALSE;
    }
    idx = idx + 1;
}
```

pour l'instant toutes les cases OK

Pattern classiques : « pour tout »

déterminer si toutes les cases `i` telles que un `test(t[i])` est vrai.

```
int idx = 0;
int candidatReponse = TRUE;
while(id<9) {
    if(! test[i]) {
        candidatReponse=FALSE;
    }
    idx = idx + 1;
}
```

pour l'instant toutes les cases OK

basculement (définitif) du booléen

ne pas remettre à true

Pattern classiques : « pour tout »

déterminer si toutes les cases `i` telles que un `test(t[i])` est vrai.

```
int idx = 0;
int candidatReponse = TRUE;
while(id<9) {
    if(! test[i]) {
        candidatReponse=FALSE;
    }
    idx = idx + 1;
}
```

pour l'instant toutes les cases OK

basculement (définitif) du booléen

ne pas remettre à true

candidatReponse = réponse à la question

- Remarque : si 0 élément réponse TRUE

Pattern classiques : « meilleure case »

déterminer la meilleure case `i` pour `compare(t[i],t[j])`

```
int idx = 0;
int candidatReponse = t[0];
while(id<9) {
    if(compare(t[i],candidatReponse)) { // t[i] meilleur candidat
        candidatReponse = t[i];
    }
    idx = idx + 1;
}
```

Pattern classiques : « meilleure case »

déterminer la meilleure case `i` pour `compare(t[i],t[j])`

```
int idx = 0;
int candidatReponse = t[0];
while(id<9) {
    if(compare(t[i],candidatReponse)) { // t[i] meilleur candidat
        candidatReponse = t[i];
    }
    idx = idx + 1;
}
```

Pour l'instant meilleur candidat (~~0~~, ~~1~~)

Pattern classiques : « meilleure case »

déterminer la meilleure case `i` pour `compare(t[i],t[j])`

```
int idx = 0;
int candidatReponse = t[0];
while(id<9) {
    if(compare(t[i],candidatReponse)) { // t[i] meilleur candidat
        candidatReponse = t[i];
    }
    idx = idx + 1;
}
```

Pour l'instant meilleur candidat (\emptyset , \times)

remplacement du candidat

Pattern classiques : « meilleure case »

déterminer la meilleure case `i` pour `compare(t[i],t[j])`

```
int idx = 0;
int candidatReponse = t[0];
while(id<9) {
    if(compare(t[i],candidatReponse)) { // t[i] meilleur candidat
        candidatReponse = t[i];
    }
    idx = idx + 1;
}
```

Pour l'instant meilleur candidat (\emptyset , \times)

remplacement du candidat

- Remarque : pas de sens si 0 élément
- Remarque : `i` peut démarrer à 1

Fin