Introduction à la Programmation des Algorithmes

2.3. Langage C – Portée et variables locales

François Fleuret

https://fleuret.org/11x001/

UNIVERSITÉ
DE GENÈVE

Dans tout ce que nous avons vu jusque là, nous avons déclaré des variables au début de notre programme et ensuite utilisé ces variables dans la suite.

Un programme est presque toujours construit en multiples modules qui chacun manipule un grand nombre de variables, mais qui inter-opèrent via un petit nombre de variables.

Il est crucial que l'on puisse écrire un "bout de programme" en étant assuré de ne pas perturber le reste.

Avec ce que nous avons vu, si nous écrivions un programme complexe:

- nous devrions avoir un grand nombre de variables déclarées tout au début, loin de là où elles sont utilisées, et
- nous aurions envie d'utiliser à différents endroits des variables de même nom (`n`, `valeur_max`, `nb`, `somme`, etc.)

Pour résoudre ces problèmes le C, et la grande majorité des langages de programmation, offrent les mécanismes suivants:

- on peut déclarer une variable n'importe où,
- elle n'existe que dans la sous-partie du programme où elle a été déclarée,
- si une variable déclarée dans une sous-partie a le même nom qu'une variable déjà existante, elle masque cette dernière.

On appelle **portée d'une variable** la sous-partie du programme où elle existe et peut être utilisée.

C'est en général entre sa déclaration et la fermeture du bloc {} où elle se trouve. Nous verrons deux cas particuliers pour `for` et `switch`.

On peut déclarer une variable n'importe où.

```
1   #include <stdio.h>
2
3   int main(void) {
4     int n;
5
6     n = 0;
7
8     printf("C'est parti!\n");
9
10    int m;
11
12    n = 3;
13    m = 2 * n + 1;
14
15    return 0;
16  }
```

Ici une variable `m` est déclarée ligne 10, après que des expressions ont été évaluées lignes 6–8.

Une variable n'existe que dans la sous-partie du programme où elle a été déclarée.

```
1   #include <stdio.h>
2
3   int main(void) {
4     int n = 0, s = 0;
5
6     while(n < 10) {
7       int m;
8       m = 2 * n + 1;
9       s = s + m;
10      n++;
11    }
12
13    int m;
14    m = 2 * n + 1;
15
16    return 0;
17  }
```

La variable `m` déclarée ligne 7 a une portée qui s'arrête ligne 11, et on peut donc déclarer une **autre** variable avec le même identifiant ligne 13.

En particulier, comme une variable cesse d'exister en dehors de sa portée, on ne peut pas y faire référence ailleurs.

```c
#include <stdio.h>

int main(void) {
  {
    int a = 3;
    printf("a=%d\n", a);
  }

  printf("a=%d\n", a);

  return 0;
}
```

```
clang test.c
test.c:9:20: error: use of undeclared identifier 'a'
  printf("a=%d\n", a);
                   ^
1 error generated.

Compilation exited abnormally with code 1 at Wed Aug  4 17:06:33
```

Une nouvelle variable masque une variable déjà existante ayant le même identifiant.

```
1   #include <stdio.h>
2
3   int main(void) {
4     int n = 0, s = 0;
5     int m = -1;
6
7     while(n < 10) {
8       int m;
9       m = 2 * n + 1;
10      s = s + m;
11      n++;
12    }
13
14    printf("s=%d m=%d\n", s, m);
15
16    return 0;
17  }
```

La variable `m` déclarée et initialisée ligne 5 est masquée par la variable de même nom déclarée ligne 8 dont la portée s'arrête ligne 12. La modification de cette dernière ligne 9 ne change par la valeur de celle déclarée précédemment, à laquelle il est fait référence ligne 14.

Des déclarations peuvent être faites dans des clauses imbriquées à plusieurs niveaux.

```
1   int n = 1;
2
3   printf("#1 n=%d\n", n);
4   {
5     int n = 2;
6     printf("#2 n=%d\n", n);
7     {
8       int n = 3;
9       printf("#3 n=%d\n", n);
10    }
11    printf("#4 n=%d\n", n);
12  }
13  printf("#5 n=%d\n", n);
```

affiche

Des déclarations peuvent être faites dans des clauses imbriquées à plusieurs niveaux.

```c
int n = 1;

printf("#1 n=%d\n", n);
{
  int n = 2;
  printf("#2 n=%d\n", n);
  {
    int n = 3;
    printf("#3 n=%d\n", n);
  }
  printf("#4 n=%d\n", n);
}
printf("#5 n=%d\n", n);
```

affiche

```
#1 n=1
#2 n=2
#3 n=3
#4 n=2
#5 n=1
```

Il est possible de faire des déclarations avec l'instruction `for`. La portée est alors restreinte à la clause exécutée dans la boucle.

```
1  int n = 42;
2
3  for(int n = 0; n < 3; n++) {
4    printf("#1 n=%d\n", n);
5  }
6
7  printf("#2 n=%d\n", n);
```

affiche

```
#1 n=0
#1 n=1
#1 n=2
#2 n=42
```

Une variable `n` est déclarée ligne 3, avec une portée qui va du premier `;` de la ligne 3, jusqu'à la ligne 5, et elle masque celle déclarée ligne 1.

```
1  int a = 5;
2
3  for(int n = 1; n < a; n++) {
4    int a = 300;
5    printf("n=%d\n", n);
6  }
```

affiche

```
n=1
n=2
n=3
n=4
```

La variable a à laquelle il est fait référence ligne 3 est celle déclarée ligne 1, et non pas la variable locale déclarée ligne 4.

La déclaration de variables dans un switch est possible, mais il est nécessaire de définir explicitement la portée avec {}.

```
1  switch(n) {
2  case 0:
3    printf("zero\n");
4    break;
5  case 1:
6    {
7      printf("un\n");
8      int k = 4;
9      do {
10       printf("%d\n", k);
11       k--;
12     } while(k > 0);
13   }
14   break;
15 default:
16   printf("beaucoup\n");
17   break;
18 }
```

La variable déclarée ligne 8 a une portée qui va jusqu'à la ligne 13.

Une variable qui n'est utilisable que dans une sous-partie du programme est dite **locale**.

Depuis le début de ce cours, toutes les variables que nous avons utilisées ont une portée limitée à la fonction `main`.

Bien que cela soit très rare, il peut être nécessaire de déclarer des variables **globales**. Il suffit de les déclarer en dehors de la fonction `main`.

```c
#include <stdio.h>

float pi = 3.14159265359;

int main(void) {
  for(int r = 1; r <= 4; r++) {
    printf("diametre = %d m surface = %f m^2\n", 2 * r, pi * r * r);
  }

  return 0;
}
```

affiche

```
diametre = 2 m surface = 3.141593 m^2
diametre = 4 m surface = 12.566371 m^2
diametre = 6 m surface = 28.274334 m^2
diametre = 8 m surface = 50.265484 m^2
```

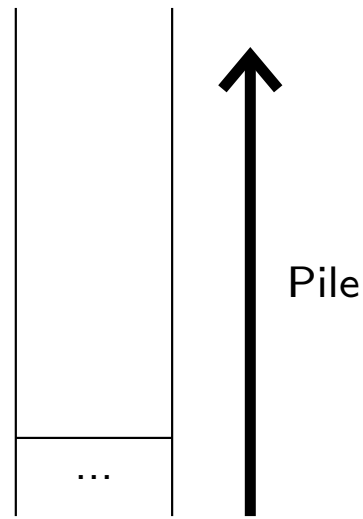
Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

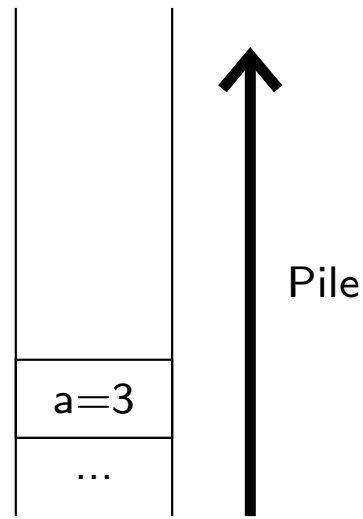
```
int main(void) {
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

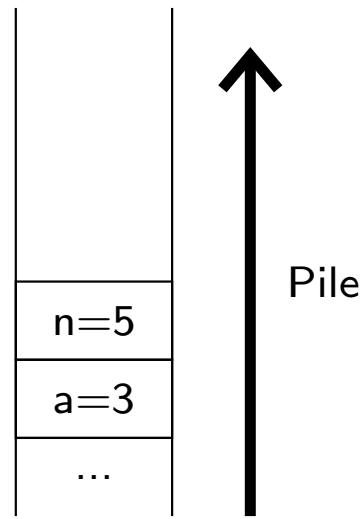
```c
int main(void) {
  int a = 3;
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

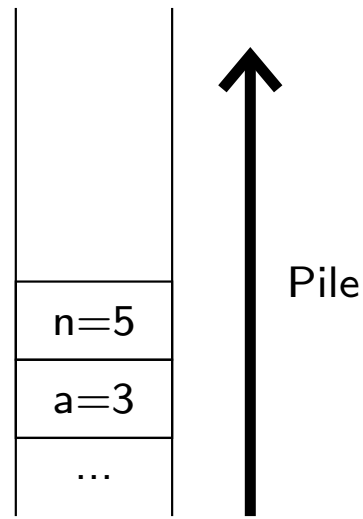
```c
int main(void) {
  int a = 3;
  int n = 5;
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

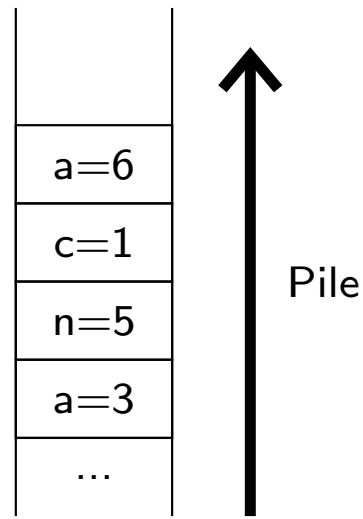
```c
int main(void) {
  int a = 3;
  int n = 5;

  if(n > 3) {
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

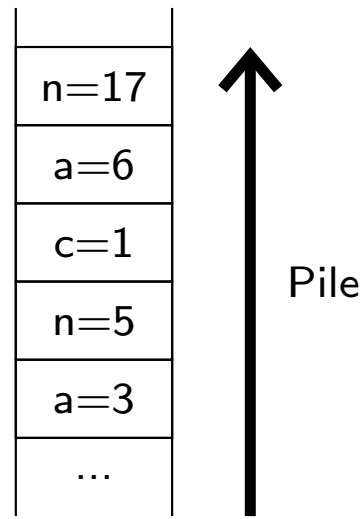
```c
int main(void) {
  int a = 3;
  int n = 5;
  if(n > 3) {
     int c = 1, a = n + 3;
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

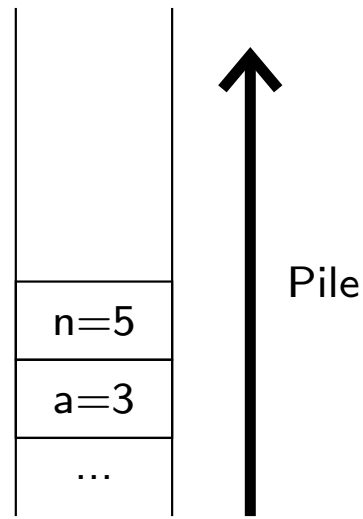
```c
int main(void) {
  int a = 3;
  int n = 5;
  if(n > 3) {
     int c = 1, a = n + 3;
     int n = 17;
     printf("a=%d c=%d n=%d\n", a, c, n);
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

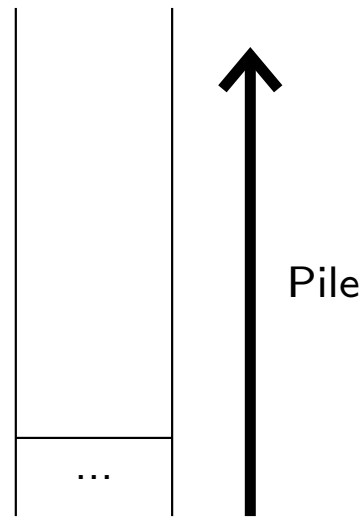
```c
int main(void) {
  int a = 3;
  int n = 5;
  if(n > 3) {
     int c = 1, a = n + 3;
     int n = 17;
     printf("a=%d c=%d n=%d\n", a, c, n);
  }
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

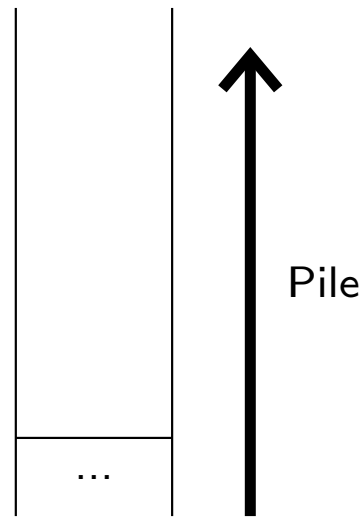
```c
int main(void) {
  int a = 3;
  int n = 5;

  if(n > 3) {
     int c = 1, a = n + 3;
     int n = 17;
     printf("a=%d c=%d n=%d\n", a, c, n);
  }

  return 0;
}
```



Les variables locales sont stockées en mémoire dans une zone que l'on appelle la **pile**. À chaque fois qu'une nouvelle variable est déclarée, le compilateur réserve la zone en haut de la pile.

Un identifiant fait référence à la variable la plus haute dans la pile qui a ce nom.

```c
int main(void) {
  int a = 3;
  int n = 5;

  if(n > 3) {
     int c = 1, a = n + 3;
     int n = 17;
     printf("a=%d c=%d n=%d\n", a, c, n);
  }

  return 0;
}
```



Il peut arriver qu'un programme essaye d'allouer trop de variables sur la pile, ce qui entraîne le fameux "stack overflow."

Quelques remarques:

- On peut déclarer une variable aussitôt que possible dans la clause où elle est utile, ou au contraire juste avant de l'utiliser. Les deux choix sont valides, mais le style strict du C "ANSI" impose de les déclarer dès que possible dans une clause, avant toute opération.
- Il est important d'utiliser des noms de variables qui aident à comprendre leurs rôles (`prix_unitaire`, `valeur_maximum`, etc.)
- Si vous avez besoin de beaucoup de variables locales dans une clause, c'est sans doute que le programme pourrait être mieux organisé, en particulier avec des fonctions. Nous y reviendrons.

```c
int n = 5;

for(int n = 0; n < 3; n++) {
  printf("n=%d\n", n);
}

for(n = 0; n < 3; n++) {
  printf("n=%d\n", n);
}

printf("n=%d\n", n);
```

affiche

```c
int n = 5;

for(int n = 0; n < 3; n++) {
  printf("n=%d\n", n);
}

for(n = 0; n < 3; n++) {
  printf("n=%d\n", n);
}

printf("n=%d\n", n);
```

affiche

```
n=0
n=1
n=2
n=0
n=1
n=2
n=3
```

```c
int a = 1;

{
  int a = 2;
  {
    a = 3;
    printf("a=%d\n", a);
  }
  printf("a=%d\n", a);
}

printf("a=%d\n", a);
```

affiche

```c
int a = 1;

{
  int a = 2;
  {
    a = 3;
    printf("a=%d\n", a);
  }
  printf("a=%d\n", a);
}

printf("a=%d\n", a);
```

affiche

```
a=3
a=3
a=1
```

```c
int main(void) {
  int k = 0;

  do {
    int k = 3;
    for(int n = 0; n < 10; n++) {
      k += n;
    }
  } while(k < 10);

  printf("k=%d\n", k);

  return 0;
}
```

affiche

```c
int main(void) {
  int k = 0;

  do {
    int k = 3;
    for(int n = 0; n < 10; n++) {
      k += n;
    }
  } while(k < 10);

  printf("k=%d\n", k);

  return 0;
}
```

n'affiche rien car il ne s'arrête jamais.

Fin