3 - Les fonctions en C

Qu'est ce qu'une fonction

- Une fonction associe un nom à un bloc de code
- On peut appeler la fonction pour executer ce bloc de code :
- Nous avons déjà utilisé printf et scanf qui sont des fonctions

Arguments d'une fonction

- Une fonction peut accepter des arguments ce sont des variables dont la valeur sera copiée et utilisée par la fonction. Pour cela on passe les arguments entre parenthèses après le nom de la fonction.
- Par exemple lorsqu'on utilise la fonction *printf* on passe en argument une chaine de format ainsi qu'une serie d'objets à afficher ;

```
printf("Bonjour %s tu as %d ans\n",
 name, age);
```

Arguments d'une fonction

- Une fonction peut renvoyer une valeur, l'appel à la fonction se comportera comme n'importe quelle expression
- Par exemple on peut utiliser la fonction strlen pour connaitre la longueur d'une chaîne de charactères ou la fonction pow pour calculer des puissances.

```
int longueur_bonjour = strlen("bonjour"
 );
float trois_puissance_quatre = pow(3,
 4);
```

Déclaration d'une fonction

- Comme pour les variables, avant de l'utiliser, on doit déclarer le type d'une fonction
- Plus précisément, il faut déclarer :
  - Le type de la valeur renvoyée par la fonction
  - Le nom de la fonction
  - Les types et noms des arguments de la fonction
- Par exemple les fonctions strlen et pow sont déclarées ainsi :

```
int strlen(char* text);
double pow(double x, double y);
```

Définition d'une fonction

- Une fois la fonction déclarée, nous pouvons définir son comportement.
- Pour cela on écrit entre accolades le code exécuté par la fonction
- On peut utiliser le mot clé return pour renvoyer une valeur.
- Par exemple une fonction add qui ajoute 3 entiers peut être définie ainsi :

```
int add(int a, int b, int c){
  return a + b + c;
}
```

Fonctionnement des arguments

- Les arguments d'une fonction sont des variables qui n'existent uniquement dans le code de celle ci.
- Par exemple dans le code ci-dessous, la variable b existe uniquement dans la fonction main et la variable a existe uniquement dans la fonction triple.

```
int triple(int a){
  return a + a + a;
}

int main(){
  int b = 5;
  printf("%d\n", triple(b));
}
```

Fonctionnement des arguments

- Lors de l'appel à une fonction, les valeurs passées sont copiées dans les arguments, il n'est donc pas possible de modifier une variable exterieure depuis une fonction.
- Par exemple dans le code ci-dessous, la valeur b n'est pas modifiée par la fonction ajoute.

```
int ajoute(int a){
  a = 6;
  return a + 5;
}

int main(){
  int b = 5;
  int c = tiple(b);
  printf("%d %d\n", b, c);
}
```

Fonctionnement des arguments

/!\Attention Les variables déclarées dans différentes fonctions sont des variables différentes, même si elles ont le même nom.

```
int ajoute(int a){
  a = 6;
  int b = 6;
  return a + b;
}
int main(){
  int b = 5;
  int a = ajoute(b);
  printf("a = %d\n", a);
}
```

Variable statique

- Il est possible de créer une variable statique dans une fonction, cette variable existera pour toute la durée du programme.
- Cela permet de sauvegarder une valeur entre deux appels différents d'une fonction.
- Pour la déclarer, on écrit le mot clé static avant le type de la variable

```
static char c = 'k';
```

Variable statique - Exemple

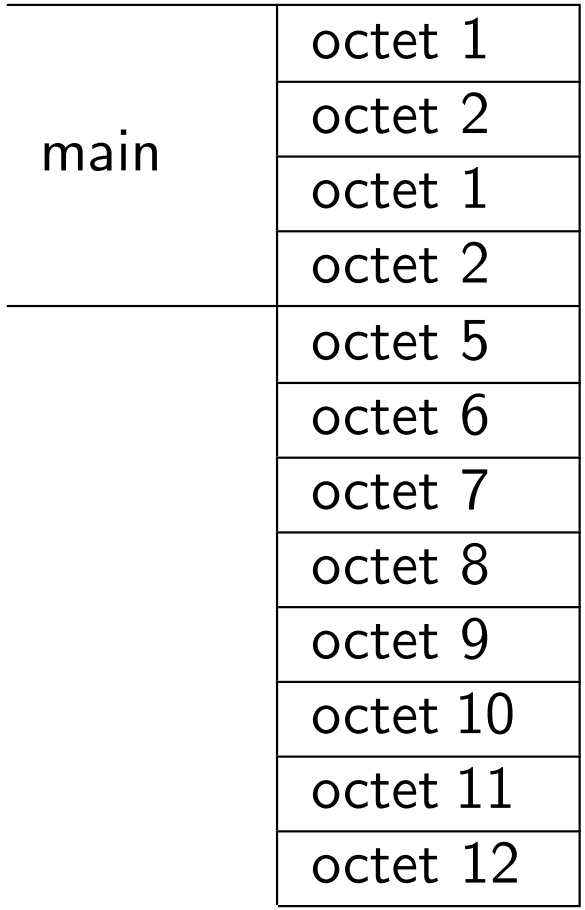
```
int fonc(int a){
    static char c = 7;
    c++;
    return a + c;
}
int main(){
    printf("%d\n", fonc(3));
    printf("%d\n", fonc(4));
}
```

Mémoire et fonctions

- Lorsqu'on appelle une fonction, un espace mémoire à la suite de ceuli utilisé par la fonction courante est alloué le temps de son execution.
- Cet espace mémoire contient assez de place pour :
  - Les arguments de la fonction.
  - Les variables locales à la fonction.
  - Des informations permettant de reprendre le programme dans son état avant l'appel de la fonction.

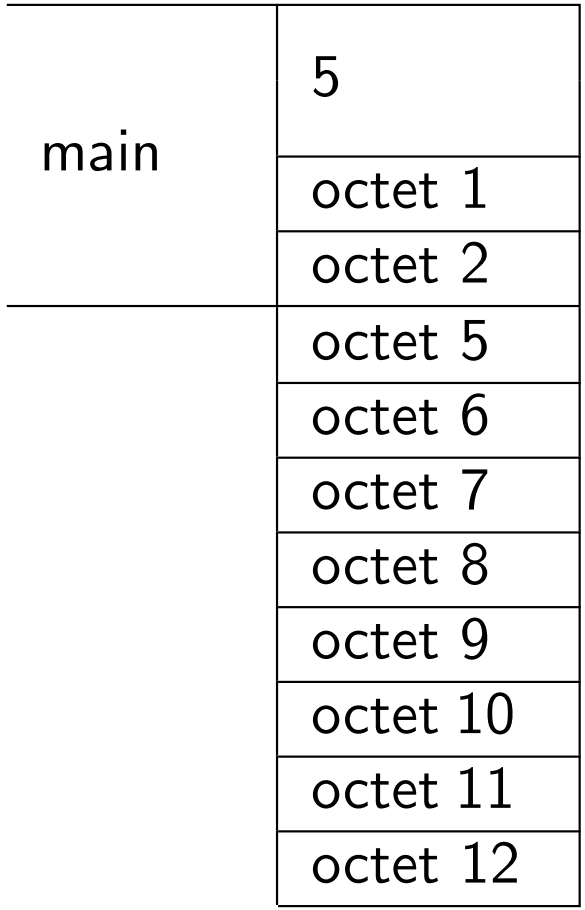
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



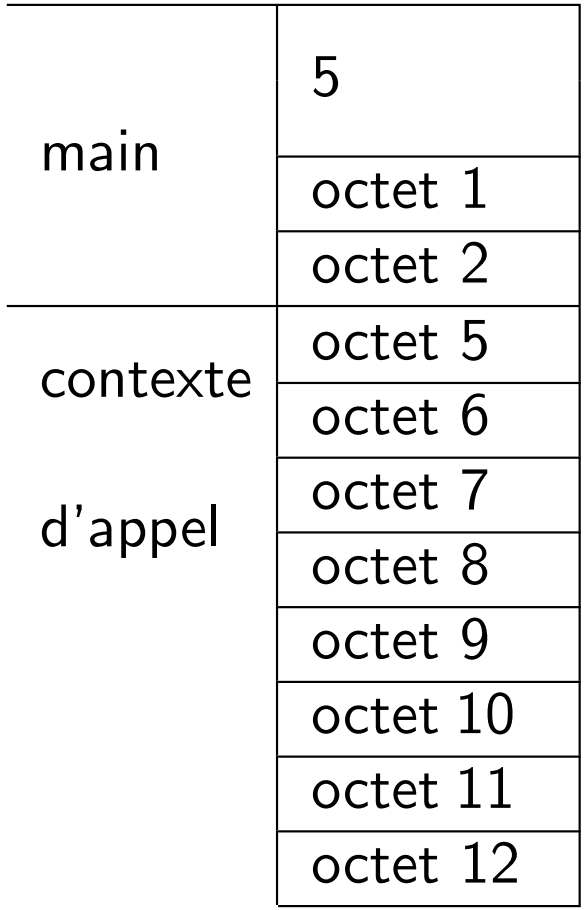
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



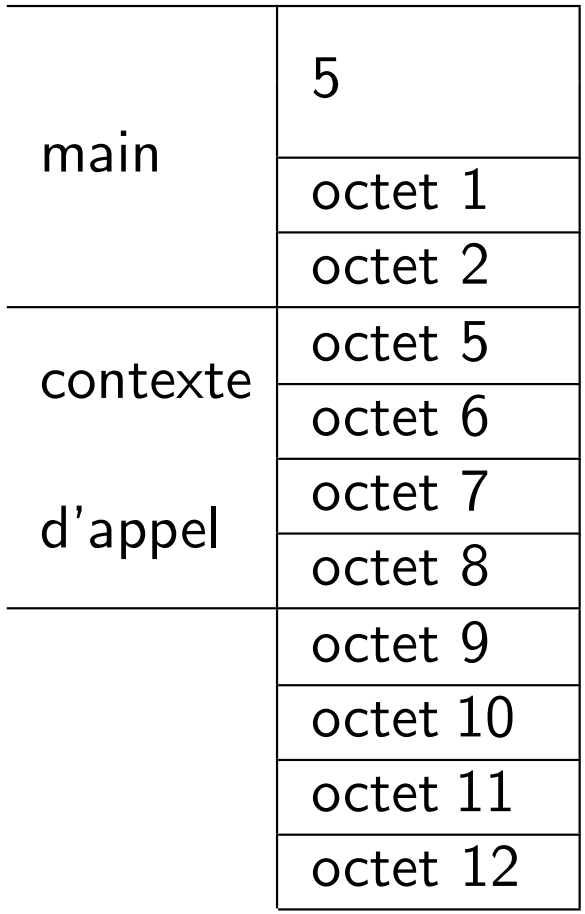
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



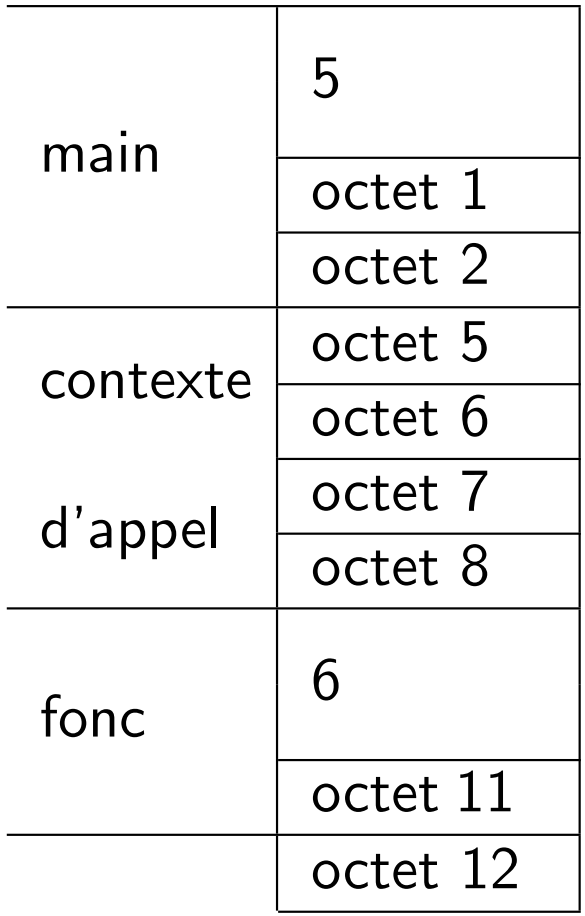
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



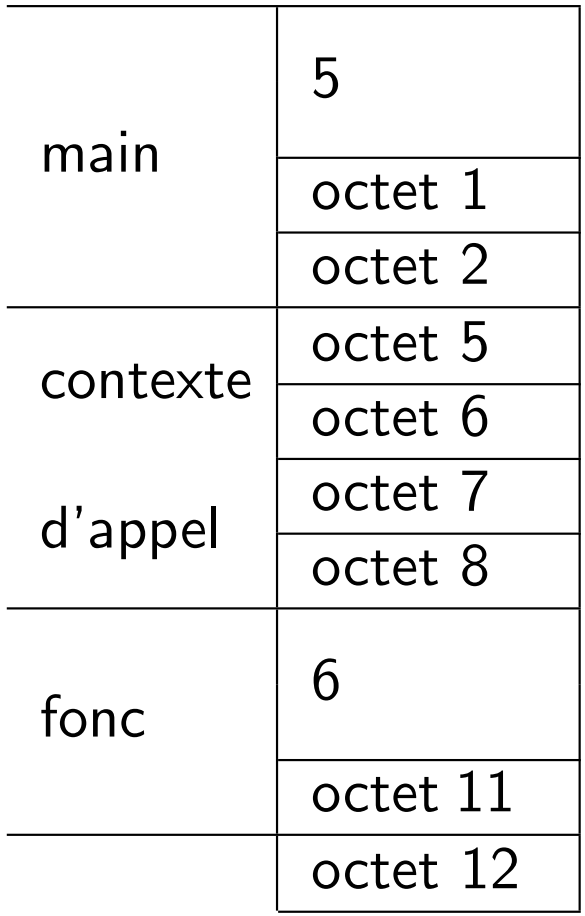
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



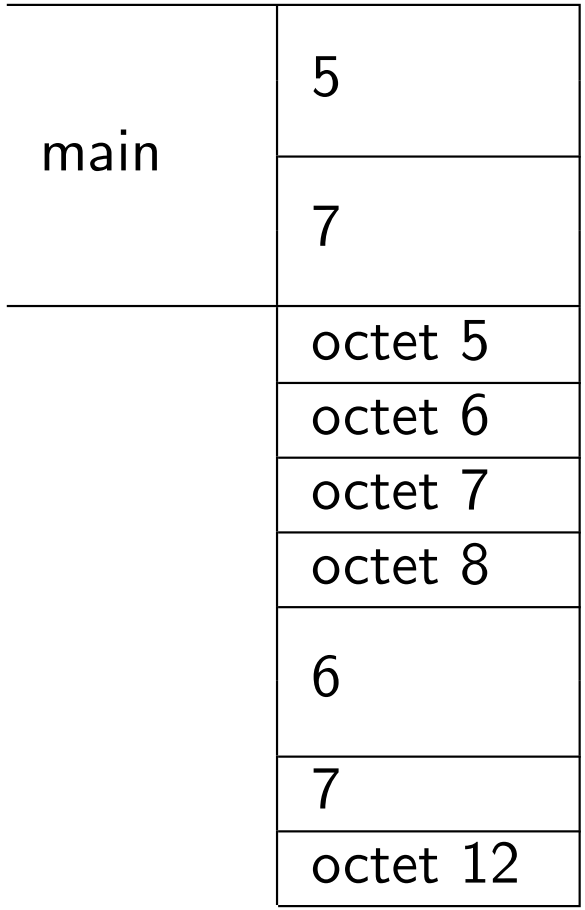
Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```

main	5
	octet 1
	octet 2
contexte d'appel	octet 5
	octet 6
	octet 7
	octet 8
fonc	6
	7
	octet 12

Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```



Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```

main	5
	7
contexte d'appel	octet 5
	octet 6
	octet 7
	octet 8
affiche	8
	7
	octet 12

Mémoire et fonction (simplifié)

```
int fonc(int a){
    char c = 7;
    return a + c;
}
int main(){
    int b = 5;
    int a = fonc(b + 1);
    affiche(a + 1);
}
```