



Bases de la programmation

Séance 4

Procédures et fonctions

Sébastien Combéfis

mercredi 1 octobre 2014



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels du cours précédent

- Le préprocesseur : **#include** et **#define**
- Définition de constante
- Définir un nouveau type avec **typedef**
- Tableau à une dimension

Déclaration, initialisation, accès par indice

- Algorithme : entrée, sortie, effet de bord
- Algorithmes sur tableau

Afficher, réinitialiser, rechercher un élément

Représentation des flottants en binaire I

- Un **flottant** est stocké sur 32 bits (IEEE 754)
 - 1 bit de signe
 - 8 bits pour l'exposant
 - 23 bits pour la mantisse
- Le nombre représenté est

$$(-1)^S \times (1.M) \times 2^{E-127}$$



Représentation des flottants en binaire III

- L'**exposant** va de -126 à 127 en forme **normalisée**

- $E = 0$ pour la forme **dénormalisée**

Représente 0 si $M = 0$ ou $(-1)^S \times (0.M) \times 2^{-126}$ sinon

- $E = 255$ pour les **nombres spéciaux**

Représente $\pm\infty$ si $M = 0$ ou NaN sinon

Procédure et fonction

- Les **procédures et fonctions** sont un bloc de code qu'on peut rappeler facilement plusieurs fois
 - Identifié par un **nom**
 - Pouvant recevoir des **paramètres**
 - Pouvant renvoyer un **résultat**

Procédure sans paramètres

```
void nom_procedure()
```

- **void** signale que c'est une **procédure**
- `nom_procedure` sert à identifier la procédure
- Une procédure représente une **action à exécuter**

Procédure sans paramètres

```
1  #include <stdio.h>
2
3  // Procédure qui affiche Hello!
4  void sayHello()
5  {
6      printf ("Hello!\n");
7  }
8
9  // Fonction principale
10 int main()
11 {
12     sayHello();
13
14     return 0;
15 }
```

Procédure avec paramètres

```
void nom_procedure (liste_parametres)
```

- Les paramètres permettent de communiquer des informations à la procédure appelée
- Un paramètre est décrit par deux éléments
 - Un type (**int**, **char**, **float**...)
 - Un nom
- Le paramètre n'existe que dans le corps de la procédure

Procédure avec paramètres

```
1  #include <stdio.h>
2
3  // Procédure qui compte de 1 jusque max
4  void countTo (int max)
5  {
6      int i;
7      for (i = 1; i <= max; i++)
8      {
9          printf ("%d\n", i);
10     }
11 }
12
13 // Fonction principale
14 int main()
15 {
16     int v = 2;
17     countTo (v);
18
19     return 0;
20 }
```

Procédure avec paramètres

main

```
1 int main() ←  
2 {  
3     int v = 2;  
4     countTo (v);  
5  
6     return 0;  
7 }
```

Écran

Procédure avec paramètres

main

`int v : 2`

```
1 int main()  
2 {  
3   int v = 2;  
4   countTo (v);  
5  
6   return 0;  
7 }
```



Écran

Procédure avec paramètres

countTo	
main	<code>int v : 2</code>

1
2
3
4
5
6
7

```
int main()
```

```
{
```

```
    int v = 2;
```

```
    countTo (v);
```

```
    return 0;
```

```
}
```

Écran

Procédure avec paramètres

countTo	<code>int max : 2</code>
main	<code>int v : 2</code>

```
1 void countTo (int max) ←
2 {
3     int i;
4     for (i = 1; i <= max; i++)
5     {
6         printf ("%d\n", i);
7     }
8 }
```

Écran

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : -</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i; ←
4     for (i = 1; i <= max; i++)
5     {
6         printf ("%d\n", i);
7     }
8 }
```

Écran

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : 1</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i;
4     for (i = 1; i <= max; i++) ←
5     {
6         printf ("%d\n", i);
7     }
8 }
```

Écran

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : 1</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i;
4     for (i = 1; i <= max; i++)
5     {
6         printf ("%d\n", i); ←
7     }
8 }
```

Écran

1\n

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : 2</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i;
4     for (i = 1; i <= max; i++) ←
5     {
6         printf ("%d\n", i);
7     }
8 }
```

Écran

1\n

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : 2</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i;
4     for (i = 1; i <= max; i++)
5     {
6         printf ("%d\n", i); ←
7     }
8 }
```

Écran

1\n2\n

Procédure avec paramètres

countTo	<code>int max : 2</code> <code>int i : 3</code>
main	<code>int v : 2</code>

```
1 void countTo (int max)
2 {
3     int i;
4     for (i = 1; i <= max; i++) ←
5     {
6         printf ("%d\n", i);
7     }
8 }
```

Écran

1\n2\n

Procédure avec paramètres

main

`int v : 2`

```
1 int main()  
2 {  
3     int v = 2;  
4     countTo (v);  
5  
6     return 0;  
7 }
```



Écran

1\n2\n

Prototype

```
1  #include <stdio.h>
2
3  void addTVA (float , int );
4
5  int main()
6  {
7      addTVA (9.99 , 21);
8
9      return 0;
10 }
11
12 void addTVA (float price , int tva)
13 {
14     printf ("Le prix est %f\n", price * (1 + tva / 100.0));
15 }
```

- Permet de mettre les procédures/fonctions sous la main

Prototype

```
1  #include <stdio.h>
2
3  void addTVA (float , int );
4
5  int main()
6  {
7      addTVA (9.99 , 21);
8
9      return 0;
10 }
11
12 void addTVA (float price , int tva)
13 {
14     printf ("Le prix est %f\n", price * (1 + tva / 100.0));
15 }
```

- Type identique à la procédure/fonction associée

Prototype

```
1 #include <stdio.h>
2
3 void addTVA ( float , int );
4
5 int main ()
6 {
7     addTVA (9.99 , 21);
8
9     return 0;
10 }
11
12 void addTVA ( float price , int tva )
13 {
14     printf ( "Le prix est %f\n" , price * (1 + tva / 100.0);
15 }
```

- Nom identique à la procédure/fonction associée

Prototype

```
1 #include <stdio.h>
2
3 void addTVA (float, int);
4
5 int main()
6 {
7     addTVA (9.99, 21);
8
9     return 0;
10 }
11
12 void addTVA (float price, int tva)
13 {
14     printf ("Le prix est %f\n", price * (1 + tva / 100.0));
15 }
```

- On peut se limiter à indiquer les types des paramètres

Fonction sans paramètres

```
type_retour nom_fonction()
```

- Renvoie une valeur du type défini (**int**, **char**, **float**...)
- `nom_fonction` sert à identifier la fonction
- Une fonction représente une **opération à calculer**

Fonction sans paramètres

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int randomDice();
5
6  // Fonction principale
7  int main()
8  {
9      srand();
10     printf ("Le dé affiche %d\n", randomDice());
11
12     return 0;
13 }
14
15 // Fonction qui lance un dé à six faces
16 int randomDice()
17 {
18     int diceValue = (rand() % 6) + 1;
19     return diceValue;
20 }
```

Instruction return

- L'instruction **return** quitte la fonction en renvoyant un résultat
- Le type renvoyé par le **return** doit être le même que celui déclaré pour la fonction
- Un **appel de fonction** est une expression

Fonction avec paramètres

```
type_retour nom_fonction (liste_parametres)
```

- Les **paramètres** permettent de communiquer des informations à la fonction appelée
- Un paramètre est décrit par deux éléments
 - Un type (**int**, **char**, **float**...)
 - Un nom
- Le paramètre n'existe que dans le corps de la fonction

Fonction avec paramètres

```
1  #include <stdio.h>
2
3  // Fonction qui fait la somme de a et b
4  int getSum (int a, int b)
5  {
6      int sum = a + b;
7
8      return sum;
9  }
10
11 // Fonction principale
12 int main()
13 {
14     int sum = getSum(2, 3);
15
16     printf ("La somme de 2 + 3 vaut %d.\n", sum);
17
18     return 0;
19 }
```

Fonction avec paramètres

main

```
1 int main() ←  
2 {  
3     int sum = getSum (2, 3);  
4  
5     printf ("La somme de 2 + 3 vaut %d.\n", sum);  
6  
7     return 0;  
8 }
```

Écran

Fonction avec paramètres

getSum	
main	<code>int sum : —</code>

```
1 int main()  
2 {  
3     int sum = getSum (2, 3); ←  
4     printf ("La somme de 2 + 3 vaut %d.\n", sum);  
5     return 0;  
6 }  
7  
8
```

Écran

Fonction avec paramètres

getSum	<code>int a : 2</code> <code>int b : 3</code>
main	<code>int sum : —</code>

```
1 int getSum (int a, int b) ←  
2 {  
3     int sum = a + b;  
4  
5     return sum;  
6 }
```

Écran

Fonction avec paramètres

getSum	<code>int a : 2</code> <code>int b : 3</code> <code>int sum : 5</code>
main	<code>int sum : —</code>

```
1 int getSum (int a, int b)
2 {
3     int sum = a + b; ←
4
5     return sum;
6 }
```

Écran

Fonction avec paramètres

getSum	<code>int a : 2</code> <code>int b : 3</code> <code>int sum : 5</code>
main	<code>int sum : —</code>

```
1 int getSum (int a, int b)
2 {
3     int sum = a + b;
4
5     return sum; ←
6 }
```

Écran

Fonction avec paramètres

main

int sum : 5

```
1 int main()  
2 {  
3     int sum = getSum (2, 3); ←  
4     printf ("La somme de 2 + 3 vaut %d.\n", sum);  
5  
6     return 0;  
7  
8 }
```


Écran

Fonction avec paramètres

main

int sum : 5

```
1 int main()  
2 {  
3     int sum = getSum (2, 3);  
4  
5     printf ("La somme de 2 + 3 vaut %d.\n", sum);  
6  
7     return 0;  
8 }
```




Écran

La somme de 2 + 3 vaut 5.\n

Fonction avec paramètres

main

int sum : 5

```
1 int main()  
2 {  
3     int sum = getSum (2, 3);  
4  
5     printf ("La somme de 2 + 3 vaut %d.\n", sum);  
6  
7     return 0;   
8 }
```

Écran

La somme de 2 + 3 vaut 5.\n