C : calculs et structures de contrôle

Crédits

- Ce cours de OpenClassRoom

Crédits

- Ce cours de OpenClassRoom
- Un autre de Zeste de savoir

Déclaration

Voici la syntaxe :

```
1  type nomFonction(parametres)
2 {
3     // corps de la fonction
4 }
```

On indique donc :

type : le type des éléments de sortie. Le type de retour peut être par exemple `char`, `int`, `double` etc... ou rien du tout : `void`. Dans ce dernier cas, on ne parle pas de *fonction* mais de *procédure*.

Déclaration

Voici la syntaxe :

```
1  type nomFonction(parametres)
2 {
3     // corps de la fonction
4 }
```

On indique donc :

type : le type des éléments de sortie. Le type de retour peut être par exemple `char`, `int`, `double` etc... ou rien du tout : `void`. Dans ce dernier cas, on ne parle pas de *fonction* mais de *procédure*.

nom : le nom de la fonction. Par convention, il doit être indicatif de la raison pour laquelle on a créé cette fonction. Ainsi, une fonction qui évalue la factorielle d'un entier sera-t-elle appelée `factorielle` ou `f` plutôt que `dupont`.

Déclaration

Voici la syntaxe :

```
1  type nomFonction(parametres)
2  {
3      // corps de la fonction
4  }
```

On indique donc :

type : le type des éléments de sortie. Le type de retour peut être par exemple `char`, `int`, `double` etc... ou rien du tout : `void`. Dans ce dernier cas, on ne parle pas de *fonction* mais de *procédure*.

nom : le nom de la fonction. Par convention, il doit être indicatif de la raison pour laquelle on a créé cette fonction. Ainsi, une fonction qui évalue la factorielle d'un entier sera-t-elle appelée `factorielle` ou `f` plutôt que `dupont`.

parametres : les arguments qu'on passe à la fonction (zéro ou plusieurs).

Exemple

```c
double square (double d){
  return d * d;
}
```

Le mot clé `return`, comme en `Python`, interrompt le déroulement de la fonction et renvoie la valeur voulue.

Programme complet avec appel de fonction dans `main`

```c
# include <stdio.h>

double square (double d){// fonction double
  return d * d;
}

void affiche_resultat_square(double d)
{// une procédure
  printf("retour de square(%f) : %f\n",d,  square(d));
  // noter l'absence du mot clé return
}

int main()
{
  double d = 23.1;
  affiche_resultat_square(d);
  return 0;
}

```

Fonction à 4 variables

Voici un exemple de fonction à 4 variables qui renvoie la valeur d'une fonction polynôme pris en une variable `x` :

```
1 double trinome (double a, double b, double c, double x){
2   //renvoie ax^2+bx+c
3   return a*x*x + b*x + c;
4 }
```

Notion de prototype

- En C on appelle *prototype* ce qu'on désignerait par *signature* dans un autre langage : la déclaration de fonction et le type de retour ainsi que celui des arguments.

Notion de prototype

- En `C` on appelle *prototype* ce qu'on désignerait par *signature* dans un autre langage : la déclaration de fonction et le type de retour ainsi que celui des arguments.
- Pour que la fonction `main` du programme principal `monProgramme.c` ne soit pas perdue si on lui fait appeler une fonction `f` écrite ailleurs, il faut placer le prototype de `f` avant le code de `main` dans `monProgramme.c`.

Notion de prototype

- En `C` on appelle *prototype* ce qu'on désignerait par *signature* dans un autre langage : la déclaration de fonction et le type de retour ainsi que celui des arguments.
- Pour que la fonction `main` du programme principal `monProgramme.c` ne soit pas perdue si on lui fait appeler une fonction `f` écrite ailleurs, il faut placer le prototype de `f` avant le code de `main` dans `monProgramme.c`.
- Le corps lui-même de la fonction `f` peut alors être écrit après `main` ou même dans un autre fichier que dans `monProgramme.c`.

Déclaration après `main`

- Le code

```c
# include <stdio.h>

int main()
{
   double d = 23.1;
   printf("retour de square(%f) : %f\n",d,  square(d));
   return 0;
}

double square (double d){
   return d * d;
}
```

soulève une reeur à la compilation.

Déclaration après `main`

- Le code

```c
# include <stdio.h>

int main()
{
   double d = 23.1;
   printf("retour de square(%f) : %f\n",d,  square(d));
   return 0;
}

double square (double d){
   return d * d;
}
```

  soulève une reeur à la compilation.

- En effet, `square` est déclarée après `main`.

Notion de prototype (suite)

- Mais en mettant la déclaration avant `main` et le code après, tout se passe bien :

```c
# include <stdio.h>

double square (double d);// le prototype

int main()
{
   double d = 23.1;
   printf("retour de square(%f) : %f\n",d,  square(d));
   return 0;
}

double square (double d){
   return d * d;
}
```