0.1 Les fonctions et actions

0.1.1 1. Les fonctions :

0.1.2 1.1 Structure :

Par définition, nous donnons des paramètres à la fonction qui, après traitement, nous renvoie une valeur. En toute apparence les paramètres donnés à la fonction ne seront modifiés. La syntaxe en C pour les fonctions est la suivante :

```
TYPEla_fonction (TYPEparamètre_1, [...],TYPEparamètre_N)
{ /* L'acollade donne le début du programme*/
Déclaration des variables locales
Liste d'instructions
} /* L'acollade donne la fin du programme*/
```

TYPEindique le type de la valeur que va renvoyer la fonction. LesTYPEindiquent le type de paramètre_1 et paramètre_N. Ainsi la déclaration des paramètres s'effectue ici. Si l'on ne passe aucun paramètre à la fonction, alors il sera difficile d'indiquer un TYPE, pour remédier à cela on utilise le type **void**, par exemple :

```
int fonctionvide (void)
{
printf("cette fonction ne fait strictement rien\n");
}
```

Les acollades marquent le début et la fin de la fonction, elles sont indispensables.

0.1.3 1.2 return :

Une fonction doit renvoyer une valeur. Pour cela nous avons besoin de lui indiquer quel valeur elle doit rendre. On réalise cet objectif avec la commande return. return donne à la fonction la valeur désirée et arrête celle-ci (autrement dit, nous revenons à la fonction appelante) Pour vous faire comprendre son usage, je vous laisse soin de regarder ce petit exemple :

```
int signe (double x)
/*Cette fonction regarde le signe du réel x. Elle renvoie
   la valeur 0 s'il est négatif, 1 s'il est positif et
   2 s'il est nul*/
{
if (x < 0)
return(0);
else if (x == 0)
return (2);
else return(1);
}
```

Il est à noter que si la dernière instruction de la fonction n'est pas return, alors cette dernière renvoie tout de même une valeur mais qui nous est inconnue. Pour les cas d'erreur (comme une division par zéro) on utilise aussi return pour délivrer le message d'erreur.

0.1.4 1.3 Fonctions récurentes :

Créer une fonction récurente permet en général d'écrire une boucle while ou do de façon plus simple. Toute la difficulté réside dans la compréhension de la récurence autrement dit du cycle. Une fonction est récurente lorsqu'elle fait appel à elle-même. Voici un exemple de fonction récurente très connue : la fonction factoriel.

```
int factoriel (int n)
/*Cette fonction calcul le factoriel d'un entier n. Attention,
   n est supposé supérieur ou égal à 0. Si n est négatif,
   la valeur 1 sera renvoyé comme dans les cas n=0 ou n=1.*/
{
if (n > 1)
return (n*factorielle (n-1));
else return (1);
}
```

Remarque : nous pourrions effectuer un test comme **if (n < 0)** pour renvoyer un message d'erreur mais le problème est que le test sera effectué à chaque appel de la fonction alors qu'il est seulement utile de l'effectuer une unique fois. Aussi si l'on veut faire un test, on le fera avant l'appel de la fonction factoriel. Et pour vous donnez une idée de ce que cela donne sans utiliser la récurence :

```
int factoriel2 (int n)
/*Cette fonction calcul le factoriel d'un entier n positif.
   Si n est négatif elle renvoie la valeur -1.*/
{
int res = 1;
if (n < 0) return (-1);
while ( n > 1)
{res = res * n;
n-;}
return (res);
}
```

0.1.5 1.4. Fonction retournant un pointeur :

Et oui c'est possible :-) et cela peut quelques fois servir. Pour vous donnez un exemple de comment cela se passe :

```
int *maxtab (int *tab, int taille);
/*Cette fonction renvoie l'adresse du plus grand entier
   contenu dans le tableau tab.*/
{int i, *max;
max = tab;
for (i=1, i < taille, i++)
if (*(tab+i) > *max) max = tab + i;
return (max);
}
```

0.1.6 2. Les procédures :

Dans une procédure, nous lui donnons des paramètres qui sont, après traitement, modifiés. La procédure en soit ne renvoie aucune valeur. En C, il n'y a pas à proprement parler de procédure. Mais l'usage des pointeurs permet à partir de fonction de simuler des procédures. Aussi nous ne donnerons pas directement les paramètres que nous voulons modifier à la fonction, mais **leurs adresses mémoires**. Comme nous l'avons vu, une fonction doit retourner une valeur, or comme une procédure ne renvoie pas de valeur nous utiliserons le type **void** pour désigner cette absence. reprenons l'exemple de factoriel2 écrit cette fois à l'aide d'une procédre :

```
[void factoriel3 (int n, int *res)
/*Cette procédure calcul le factoriel d'un entier n positif.
  Si n est négatif elle renvoie la valeur *res = -1.*/
{
*res = 1;
if (n < 0) *res = -1;
while ( n > 1)
{*res = *res * n;
n-;}
}
```

Ce qu'il faut comprendre ici est que ***res** désigne la valeur détenue à l'adresse mémoire désigné par **res**. Pour plus d'information je vous laisse soin de consulter la page sur lespointeurs[1]. Comme vous le comprenez, en C tout est fonction. Aussi une procédure qui renverra une valeur ne sera pas choquant... D'un point de vu vocabulaire on parle indifférement de fonction pour désigner une fonction ou une procédure en C.

0.1.7 3. La fonction principale : main

Nous voici dans le coeur du programme avec **la fonction main**. Cette fonction appelée **fonction principale**, est celle qui regroupe et met en cohésion l'ensembles des autres fonctions définies. Lors de l'exécution d'un programme, c'est cette fonction qui est exécutée. **main** ne diffère en rien des autres fonctions, et donc l'ensemble des rêgles sur les fonctions lui est appliqué. Bien qu'il nous est pas toujours nécessaire que **main** nous renvoie une valeur, il faut tout de même lui indiquer un **TYPE** autre que **void** si vous ne voulez pas être insulté par gcc lors de la compilation. Cela fait simplement parti des standards. Ainsi la ligne : **void main (void)** sera considérée comme éronné par gcc (bien qu'il compilera tout de même). En la modifiant ainsi par exemple : **int main (void)** le problème sera résolu. Regardons maintenant un cas dans la pratique en écrivant une fonction main faisant appel à factoriel3 écrite plus haut :

```
int main (void)
/*Ce programme demande à l'utilisateur de rentrer un entier
  puis lui retourne le factoriel*/
{int nb, resultat;
printf ("Donnez un entier\n");
```

[1] http://www.trustonme.net/didactels/155.html

```
scanf ("%d",&nb);
factoriel3 (nb, &resultat);
printf ("le factoriel de %d est : %d\", nb, resultat);
}
```

Comme vous le remarquerez, ce programme ne saura pas calculer le factoriel pour tout n > 12. Pour vous guider sur la cause : format :-). Déclarez resultat comme double et dans le printf remplacer %d par %e pour une écriture scientifique. La difficulté à comprendre ici est la présence du signe **&** devant resultat dans la ligne **factoriel3 (nb, &resultat) ;**. Explication : resultat à été déclaré comme une variable et non l'adresse d'une variable (pointeur) or la fonction à besoin d'une adresse pour second paramètre. Par conséquent il ne faut pas lui donner la valeur **resultat** mais l'adresse mémoire où celle-ci est rangée c'est à dire **&resultat**.

0.1.8 4. Structure d'un programme :

Juste un petit rappel sur la structure d'un programme :

```
/* 1. : Liste des inclusions :*/
#include <la_librairie_standard>
#include "La_librairie_situé_dans_le_répertoire_courant"
/* 2. : Déclaration des contantes :*/
#define contante valeur
/* 3. : Déclaration des variables globales (on évite,
   c'est plutôt mauvais) :*/
TYPE nom_de_la_varaible;
/* 4. : déclaration des prototypes des fonctions (nous
   pouvons aussi effectué un fichier d'entête .h), par
   exemple :*/
factoriel3 (int,int);
/* 5. : écriture des fonctions :*/
TYPE la_fonction (déclaration_des_paramètres)
{
déclaration_des_variables_locales
liste_d_instuctions
}
/* 6. : écriture de la fonction principale main :*/
TYPE main (déclaration_des_paramètres)
{
déclaration_des_variables_locales
liste_d_instuctions
}
```

Et pour continuer dans la lancé, quelques règles pour écrire un code propre (quelque soit le langage) :

– Une fonction doit effectuer une tâche et non pas une multitude de taches

- Une fonction doit effectuer sa tâche proprement : ne pas faire d'hypothèses et étudier tous les cas particuliers
- La fonction principale n'est pas là pour calculer mais pour coordonner un ensemble
- Eviter les variables globales
- Vérifiez toujours les valeurs rentrées par les utilisateurs : c'est l'un des rares éléments que vous ne puissiez contrôler
- Commentez votre code !

«« Précédent[2] Suivant »»[3]

[2] http://www.trustonme.net/didactels/150.html
[3] http://www.trustonme.net/didactels/152.html