Les fonctions

`int f(int x){ // code de la fonction`
`  return x*2;`
`}`
 	
`main(){`
`  int x=2;`
`  int y= f(x); // appel de la fonction`
`  printf("%d=f(%d)\n",y,x);`
`  int y= f(2); // appel de la fonction`
`  printf("%d=f(%d)",y,2);`
`}`
	Une fonction en général renvoie le résultat d'un calcul ; il faudra donc associer à **int** f(int x) un **return**
	
	Le résultat affiché sera :
	4=f(2)
	4=f(2)

`double g(double x, double y){// code de la fonction`
`  return x+y*2;`
`}`
 	
`main(){`
`  double x=2;`
`  double y= g(x, 3); // appel de la fonction`
`  printf("%d=g(%d,%d)",y,x,3); // 8 = g(2 ,3)`
`}`
	Une fonction peut avoir plusieurs **arguments** (paramètres). Ici, lors de l'appel de la fonction **g(x,3)** la valeur de x(ici 2) est **recopiée dans le x local à la fonction g** et **3 est copié dans le y local à la fonction g**. **return** permet de renvoyer le résultat du calcul dans la variable **y du main**.

`void aff(int x){ / code de la fonction`
`  printf("%d",x);`
`}`
 	
`main(){`
`  int x=2;`
`  aff(x); // appel de la fonction`
`}`
	Cas particulier : si la fonction ne renvoie pas de valeur (ici elle affiche directement la valeur par exemple), utiliser le mot clé **void**, mais cela reste un cas particulier (puisqu'une fonction n'affiche rien en général)

Les pointeurs

Les **pointeurs *p** permettent de **modifier une variable n** (en passant par son **adresse &**n)

`int *p ;` `int n ;`	Déclaration d'un **pointeur** sur int Déclaration de la variable n
`p= &n ;`	p prend **l'adresse** de n
*p = 2	p pointe sur n donc équivalent à **n=2**

Les fonctions et pointeurs

`void f(int *x) {// code de la fonction par pointeur`
`  *x = *x * 2;`
`}`
`main(){`
`  int x=2;`
`  f(&x); // appel de la fonction (adresse de x)`
`  printf("%d=f(%d)",y, x );`
`}`
	Nous avons vu qu'en général le renvoi du calcul d'une fonction se faisait par le mot clé **return**. Il existe une autre solution : les **pointeurs** int *x (à condition de passer l' adresse dans le main &x)
	Le résultat affiché sera :
	4=f(2)

`int produit1(int* resultat, int x, int y) {`
`  int erreur;`
`  *resultat = x * y;`
`  if (x * y > 1000)`
`  erreur = -1;`
`  else`
`  erreur = 0;`
`  return (x * y);`
`}`
`main(){`
`  int result, erreur, x=2;`
`  erreur = f(&result ,x, 3); // appel`
`  printf("%d %d",erreur, result );`
`}`
	On peut évidemment mixer **pointeur** et mot clé **return** pour renvoyer 2 ou plusieurs valeurs. Ici on renvoie le code d'erreur (**return**) ainsi que le résultat du produit (x*y).
	Dans cet exemple, cela permet à la fonction produit1 de calculer le produit et de tester si le résultat du produit est supérieur à 1000.
	
	Le résultat affiché sera :
	0 6

Les tableaux et chaines de caractères

`int tab1[3];`
`double tab2[3] = {1,2.2,3};`
 	
`for(int i=0; tab2[i] < 3; i++){`
`  int cpt=0;`
`  if(tab2[i]>= 2 && tab2[i]<= 4)`
`  cpt++;`
`}`
	création tableau statique 3 cases mémoires
	création tableau et init des 3 cases
	
	Compter le nombre de valeurs comprises entre 2 et 4 dans le tableau tab2 (cpt =2).

`char tab3[4] = {'b','o','n','\n'};`
`char tab4[4] = "bon";// identique à tab3`
 	
`for(int i=0; tab3[i]!='\0' ; i++){`
`  int cpt=0;`
`  if(tab3[i]>='a' && tab3[i]<='z')`
`  cpt++;`
`}`
	création chaine de caractères « bon »
	création chaine de caractères « bon »
	
	Compter le nombre de lettres minuscules dans la chaine de caractères (cpt =3). Une chaine de caractères se termine par 0 (c'est ce que l'on teste dans la boucle **for**).

Les tableaux et pointeurs

Il est possible de créer un tableau avec une taille donnée par l'utilisateur (ce qui n'est pas possible pour un tableau statique).

`int *tab5;`
`int n;`
`scanf("%d",&n) ;`
`tab5 = malloc(n *sizeof(int));`
	création pointeur (tableau dynamique)
	
	scréation de n cases mémoires de type int (tab5 pointent vers les cases mémoires créées

Les 2 codes ci-dessous sont identiques

`for(int i=0;i<3;i++)` `  *(tab5+i)=i+1;`	`for(int i=0;i<3;i++)` `  tab5[i]=i+1;`	Un tableau est un pointeur sur le début du tableau donc `*(tab+i)= tab[i]`

Les fonctions et tableaux

`void modifier1(double tab[], int taille){ // code`
`  for(int i=0;i<taille;i++)`
`  tab[i]=1;`
`}`
`void modifier2(double *tab, int taille){ // code`
`  for(int i=0;i<taille;i++)`
`  tab[i]=2;`
`}`
`main(){`
`  int tab[3]={0,0,0};`
`  modifier1 (tab,2); // appel de la fonction`
`  printf("%d %d %d", tab{0],tab[1], tab[2]);`
`  modifier2 (tab,2); // appel de la fonction`
`  printf("%d %d %d", tab{0],tab[1], tab[2]);`
`}`
	Les tableaux sont des pointeurs sur le début du tableau. Donc on peut passer à une fonction soit le tableau tab[ ], soit avec le formalisme pointeur *tab et ce tableau sera modifié dans la fonction (principe du pointeur).
	
	Résultat : (seul les 2 premiers éléments du tableau sont modifiés ; taille=2)
	1 1 0
	2 2 0

`void copier(int *dest, int *src){ // code`
`  for(int i=0;src[i]!='\0';i++)`
`  dest[i]=src[i];`
`  dest[i]='\0';//ajout 0 en fin de chaine`
`}`
`main(){`
`  char chaine[]="bon";`
`  char chaine1[10];`
`  copier(chaine1,chaine); // appel`
`  printf(chaine1);`
`}`
	`void copier(int dest[], int src[])`
	ce code permet de recopier la chaine src dans dest (formalisme pointeur ou tableau)
	
	Résultat :
	bon