Cours informatique : Maîtrise de Sciences Cognitives 2003-2004

Session 5

Structures de données : manipuler des structures hétérogènes

Les structures de données `TABLEAU` et `CHAINE DE CARACTERES` sont des structures homogènes, c'est-à-dire que chaque élément de la structure est de même type. On peut avoir des tableaux d'entiers, de réels, de caractères mais pas de tableaux où le premier élément serai un entier, le second une chaîne de caractère, le troisième un réel et ainsi de suite. C'est pourtant bien utile de pouvoir manipuler une structure hétérogène de ce type pour

désigner un ensemble de propriétés différentes d'un même objet. Par exemple, un étudiant impliqué dans une expérimentation pourrait être identifié par :

- son numéro dans le panel (un entier positif),
- son prénom (une chaîne de caractère),
- son âge (un entier positif),
- le score obtenu à l'expérimentation (un réel)

Le langage C permet de regrouper ce type de données avec le type `STRUCT`.

```
struct MON_PREMIER_TYPE_DE_STRUCTURE{int NUM_PANEL, char PRENOM[20], int AGE, float SCORE}
MA_PREMIERE_STRUCTURE;
```

Le mot clé **struct** appartient au langage C ; **MON_PREMIER_TYPE_DE_STRUCTURE** est le nom que le programmeur donne à ce type de structure. Entre {}, on trouve les différents "champs" de la structure tels que le programmeur souhaite les déclarer (il peut bien sûr y avoir des champs structurés !) ; **MA_PREMIERE_STRUCTURE** est le nom de la variable correspondant à cette structure. L'accès à un champ se fait en suffixant le nom du champ au nom de la variable structurée (avec un "." entre les deux noms).

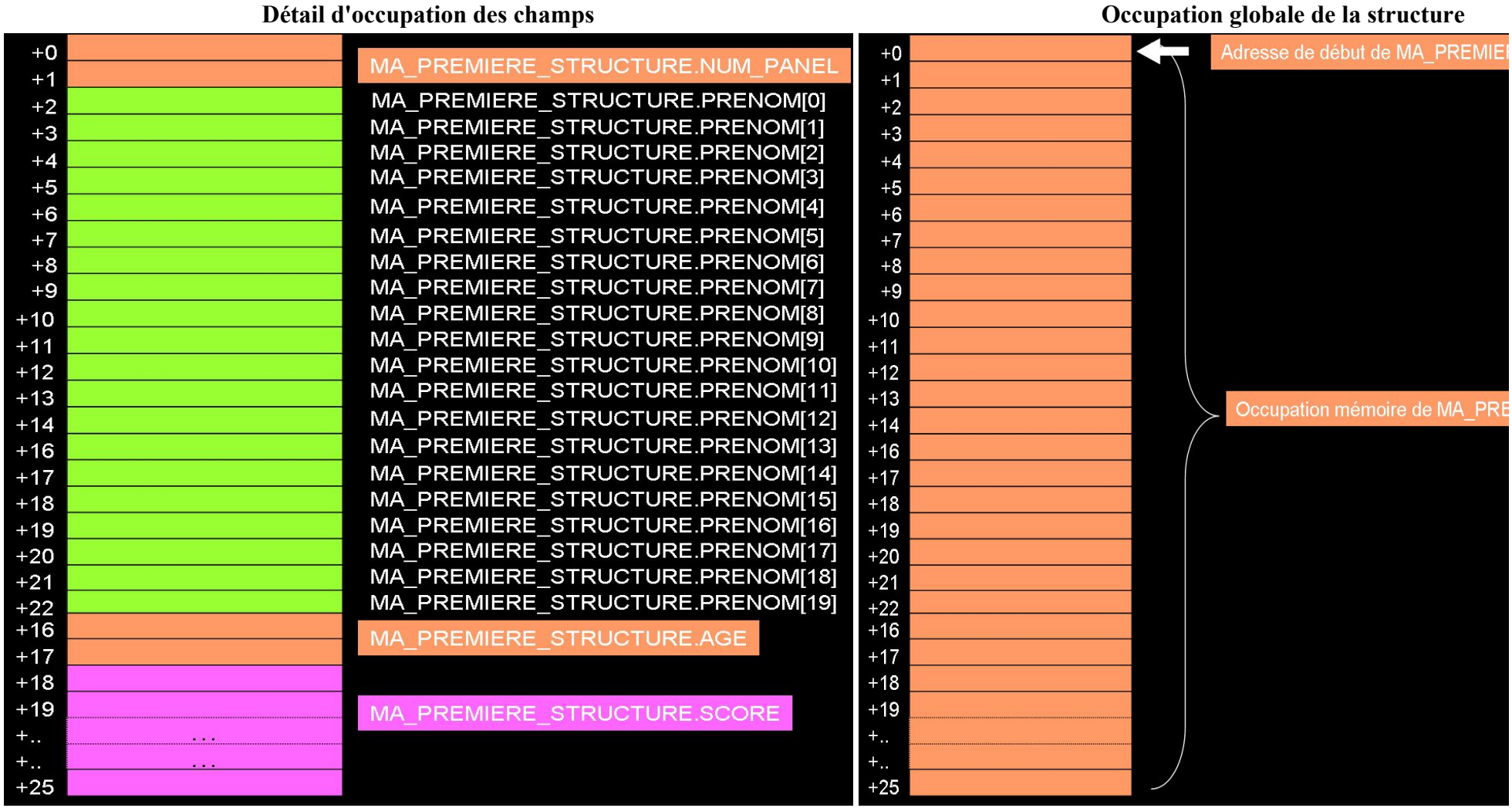
Par exemple **MA_PREMIERE_STRUCTURE.NUM_PANEL** désigne le premier champ de cette structure.

Illustration sur un code source :

```
#include <stdio.h>
int main ()
{
struct mon_premier_type_de_structure{int num_panel; char prenom[20]; int age; float score;}
ma_premiere_structure ;
ma_premiere_structure.num_panel = 12;
strcpy(ma_premiere_structure.prenom,"alain");/* la fonction strcpy(arg1,arg2) copie la chaine
contenue à l'adresse arg2 à l'adresse arg1*/
ma_premiere_structure.age = 20; /* quand on aime...*/
ma_premiere_structure.score = 18.5;
printf(" %s qui est age de %i ans appartient au panel numero %i et a reussi le score de %f /n",
ma_premiere_structure.prenom, ma_premiere_structure.age, ma_premiere_structure.num_panel,
ma_premiere_structure.score);
return 0;
```

}

Illustration de l'occupation mémoire de cet exemple :

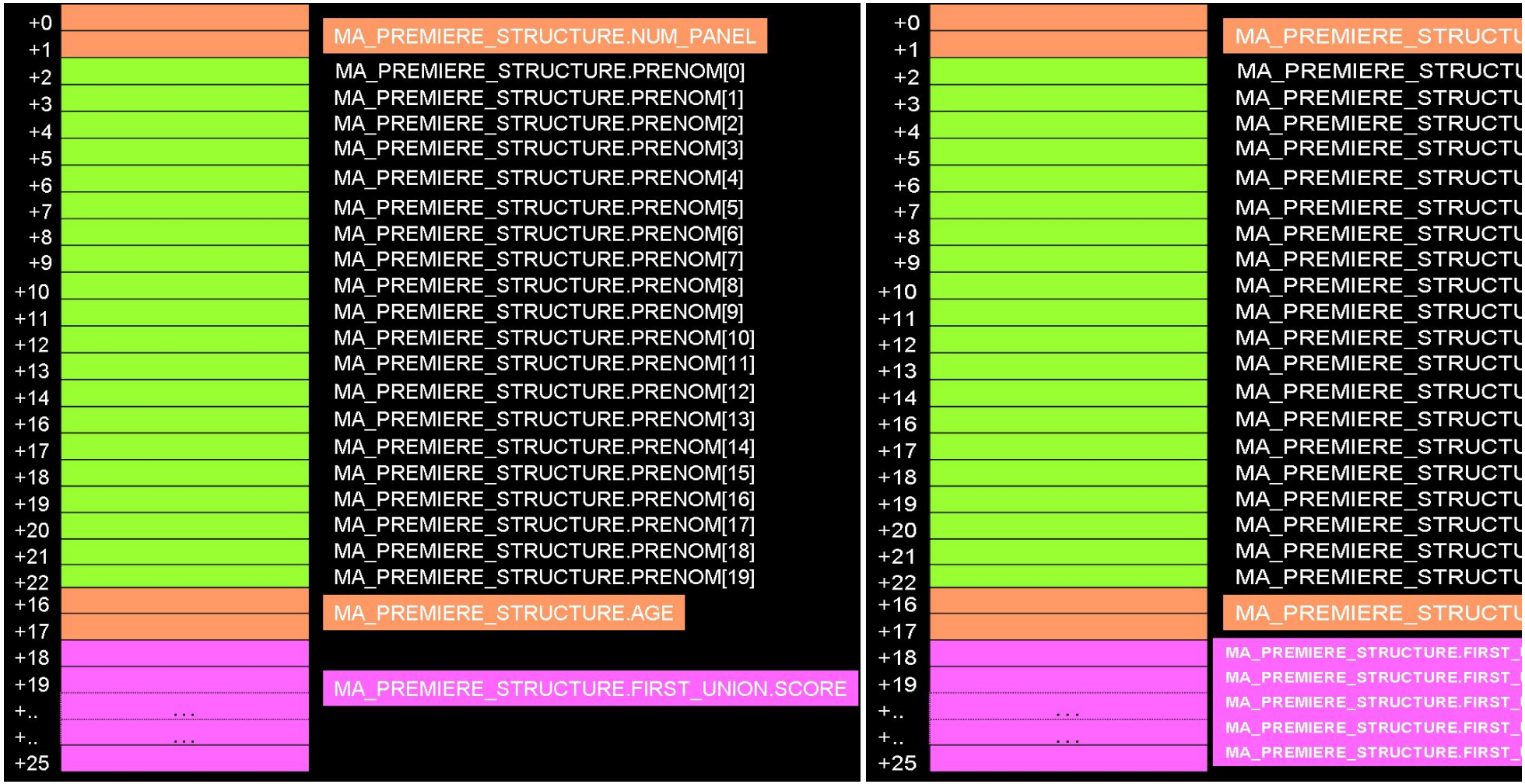



```
#include <stdio.h>
```

```
int main ()

{
struct mon_premier_type_de_structure{int num_panel; char prenom[20]; int age; union {float score;char
equiv_car[8];} first_union;} ma_premiere_structure ;
int i;

ma_premiere_structure.num_panel = 12;

strcpy(ma_premiere_structure.prenom,"alain");

ma_premiere_structure.age = 20; /* quand on aime...*/

ma_premiere_structure.first_union.score = 18.5;


printf(" %s qui est age de %i ans appartient au panel numero %i et a reussi le score de %f /n",
ma_premiere_structure.prenom, ma_premiere_structure.age, ma_premiere_structure.num_panel,
ma_premiere_structure.first_union.score);

for (i=0;i<8;i++)
printf("equiv_car [ %i ] = %x \n",i, ma_premiere_structure.first_union.equiv_car[i]);/* %x exprime
une valeur en base 16 (hexadecimal)*/

return 0;
}
```

Le schéma de la figure suivant illustre ces deux "points de vue" sur la même zone mémoire.

Détail d'occupation des champs : point de vue "first_union.score" **Détail d'occupation des champs : point de vue "first_uni**



Fonctions : modulariser un programme

Un programme en C est constitué d'une fonction "principale". Quand un programme commence à devenir complexe, ou quand on veut pouvoir réutiliser des séquences de traitement d'un programme à un autre, on a tout intérêt à "encapsuler" les instructions concernées dans une "fonction" nouvelle qui pourra être utilisée en lui fournissant un certain nombre de paramètres pour son exécution et en récupérant le résultat pour continuer le traitement. Le langage C propose déjà tout un ensemble de fonctions "préconstruites" pour traiter "les entrées-sorties" (printf, getc, putc, etc.) ou les traitements de

chaîne de caractères (strcpy, etc..).

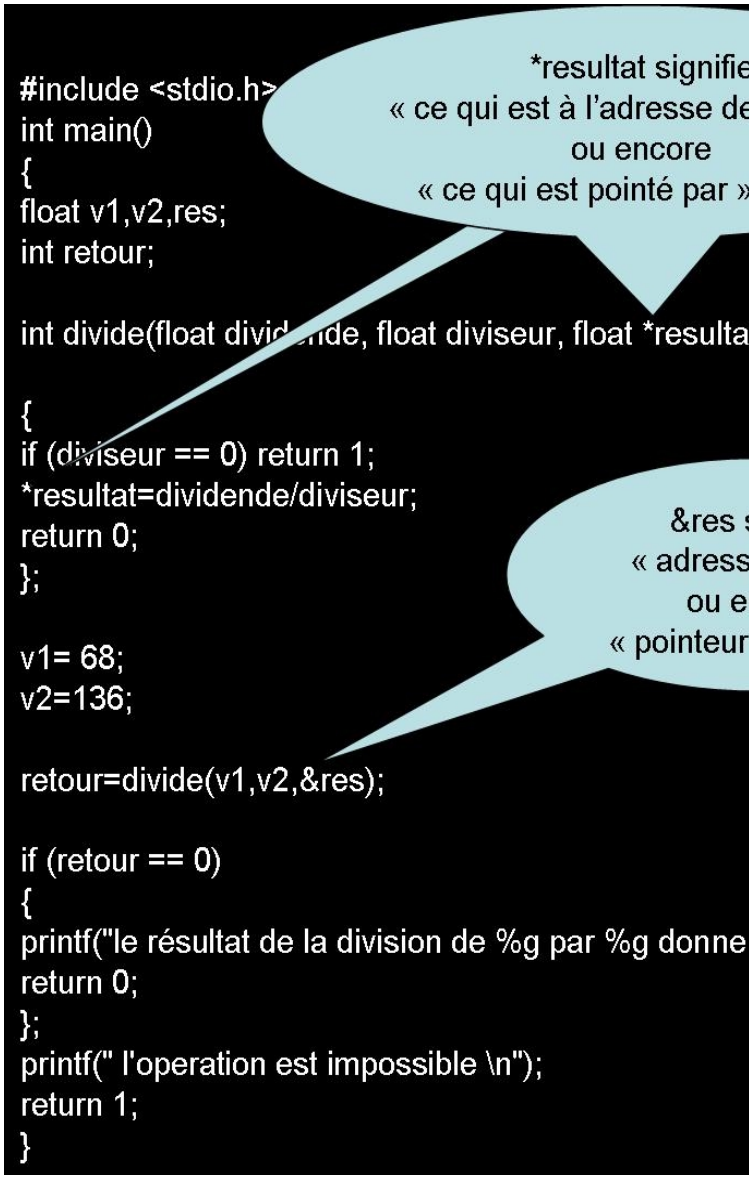
Une fonction s'écrit à peu près de la même façon que la fonction principale (voir la définition d'une fonction divide dans l'exemple suivant);:

Un premier programme avec une fonction simple

Notion de pointeur et d'adressage ind

```
#include <stdio.h>

int main()

{
float v1,v2,res;
int retour;

int divide(float dividende, float diviseur, float *resultat)

/*déclaration et définition de la fonction divide */

{
if (diviseur == 0) return 1;
*resultat=dividende/diviseur;

/* le résultat de la division est copié à l'adresse de la
variable resultat*/
return 0;
};

v1= 68;
v2=136;


retour=divide(v1,v2,&res);/* appel de la fonction divide */
```

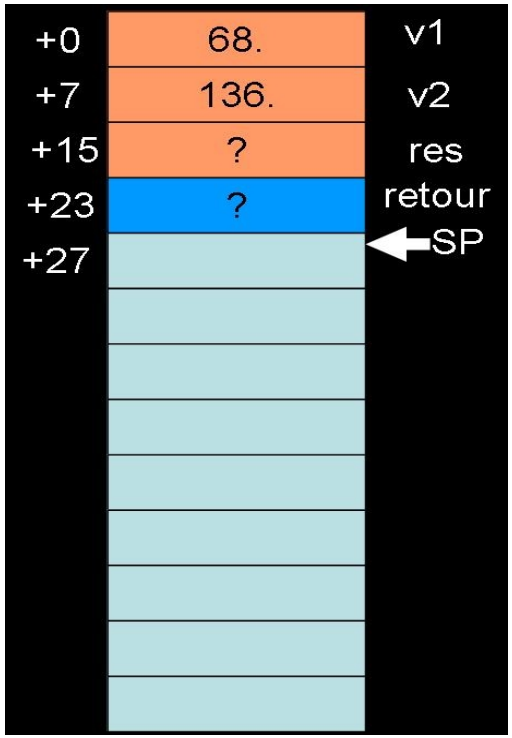
```
if (retour == 0)
{
printf("le résultat de la division de %g par %g donne %g
\n",v1,v2,res);
return 0;
};


printf(" l'operation est impossible \n");
return 1;
}
```



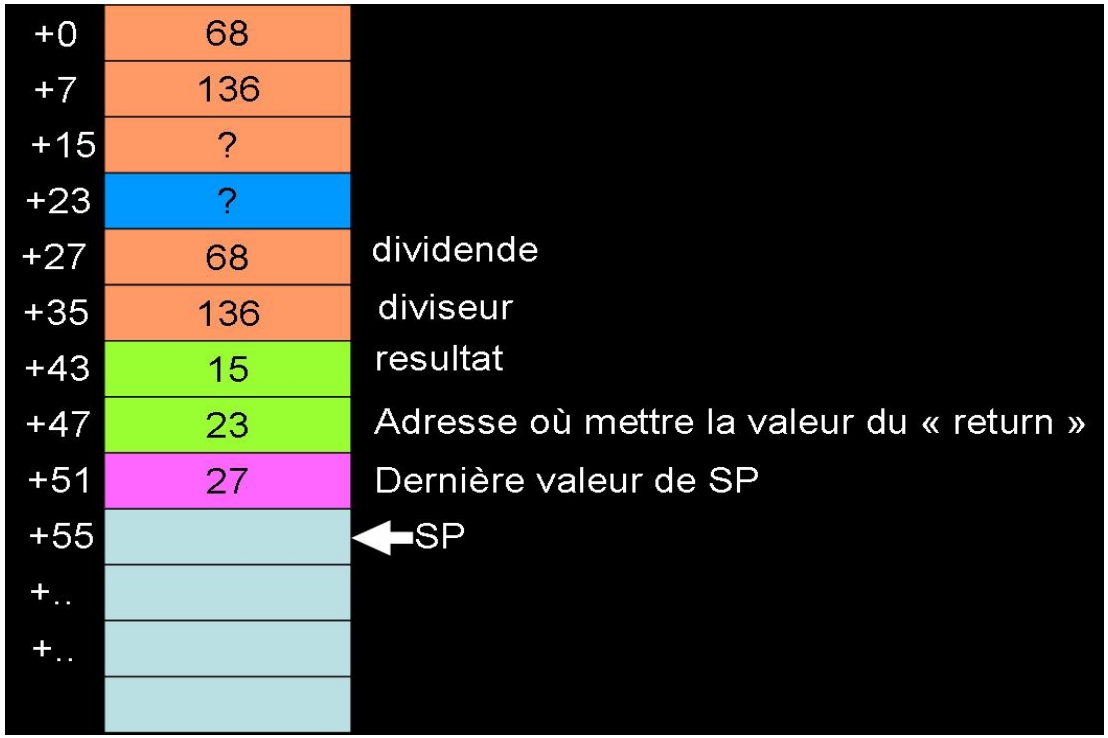
Une fonction est "étanche" aux variables de la fonction qui l'appelle. Pour que la fonction appelante ("main" dans notre exemple) communique avec la fonction appelée ("divide" dans notre exemple), il est nécessaire de passer par l'intermédiaire de "paramètres d'appels". L'appel de la fonction sera

l'occasion de réaliser la copie des paramètres d'appels dans les zones mémoires allouées pour que la fonction puisse disposer d'un espace propre pour son exécution. Le schéma suivant résume les différentes étapes de l'appel, de l'exécution et de la fin de l'exécution d'une fonction (sur l'exemple de la fonction "divide").

L'état de la pile juste avant l'appel

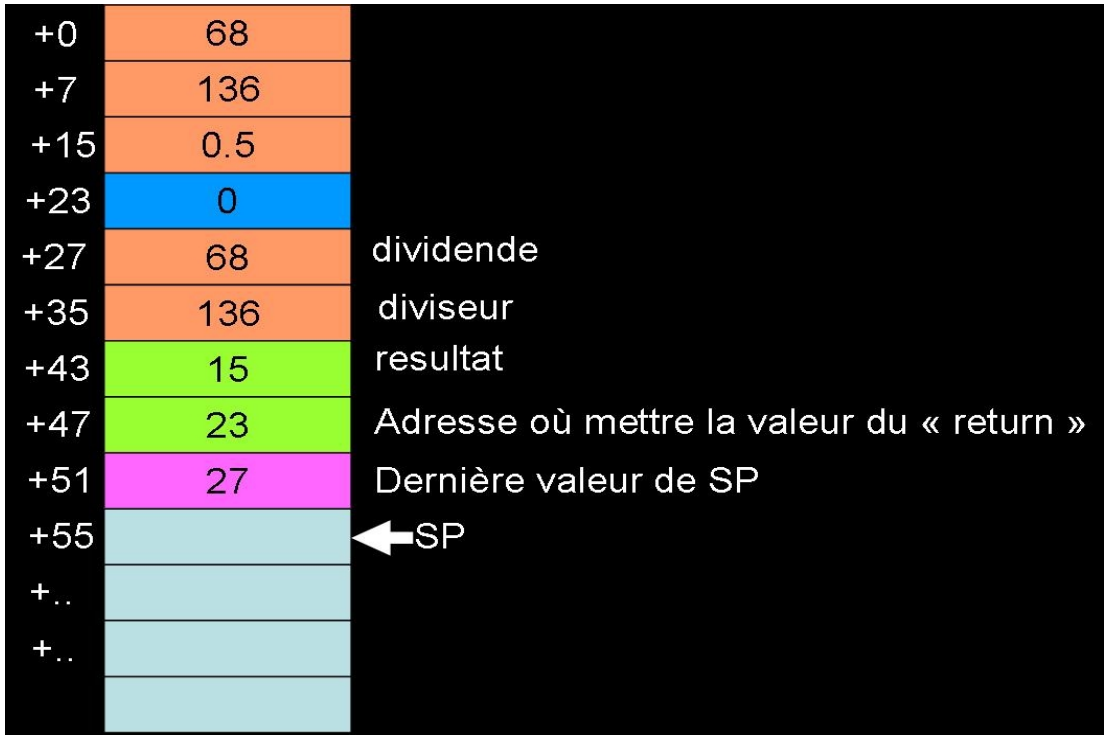


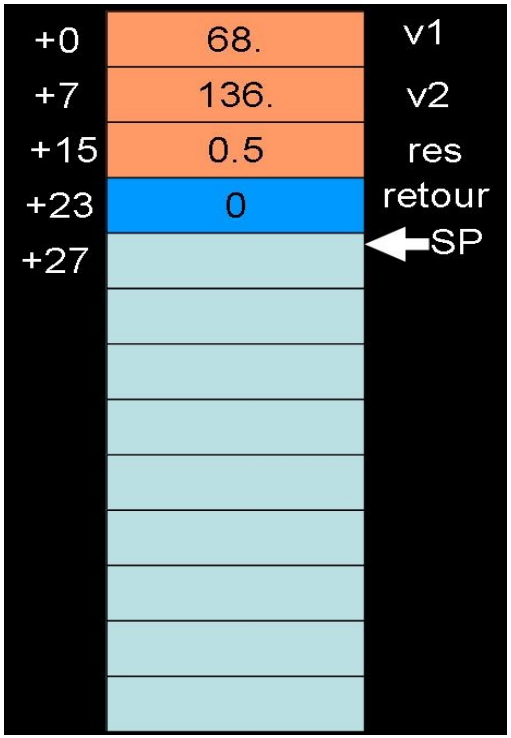
L'état de la pile au début de l'exécution de la fonction



L'état de la pile juste avant la fin de la fonction

L'état de la pile au retour dans la fonction principale





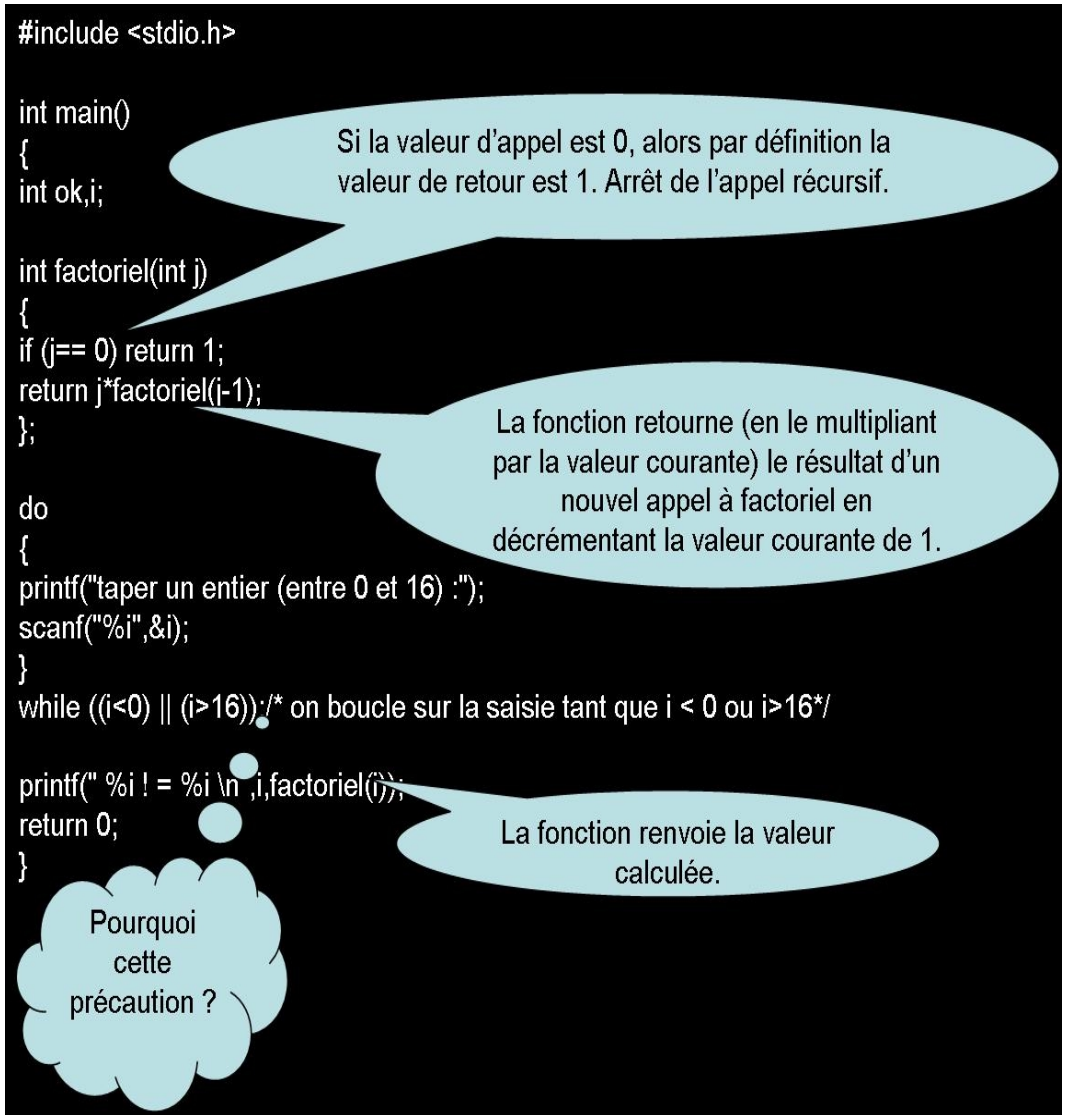
Fonctions récursives

Une fonction récursive est une fonction qui s'appelle elle-même. En pratique, la fonction porte bien le même nom et ce sera le même code qui sera exécuté, MAIS PAS SUR LES MEMES DONNEES. C'est le mécanisme de gestion de la pile tel qu'il vient d'être vu plus haut qui permet ce type de fonctionnement. Bien entendu, il convient que l'appel récursif s'arrête, permettant aux fonctions de revenir dans l'appelant et ceci jusqu'à revenir dans la fonction principale. Il est habituel de donner comme exemple de l'usage d'une fonction récursive le calcul d'une récurence. Par exemple, le calcul de "factoriel". On rappelle que n!=(n-1)!n, avec n>=0 et 0!=1 par convention (la notation n! se lit "factoriel n"). Le programme suivant est un programme permettant de calculer la factorielle d'un nombre entier entré au clavier.

Exemple de code source avec une fonction récursive A observer !

```
#include <stdio.h>

int main()
```

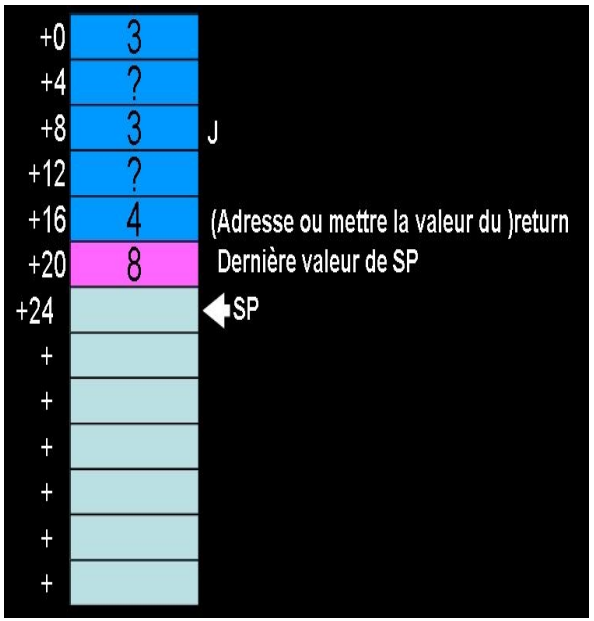
```
{
int _i;


int factoriel(int j)
{

if (j== 0) return 1;

return j*factoriel(j-1);

};

do
{
printf("taper un entier (entre 0 et 16) :");
scanf("%i",&i);
}
while ((i<0) || (i>16));/* on boucle sur la
saisie tant que i < 0 ou i>16*/

printf(" %i ! = %i \n",i,factoriel(i));

return 0;

}
```



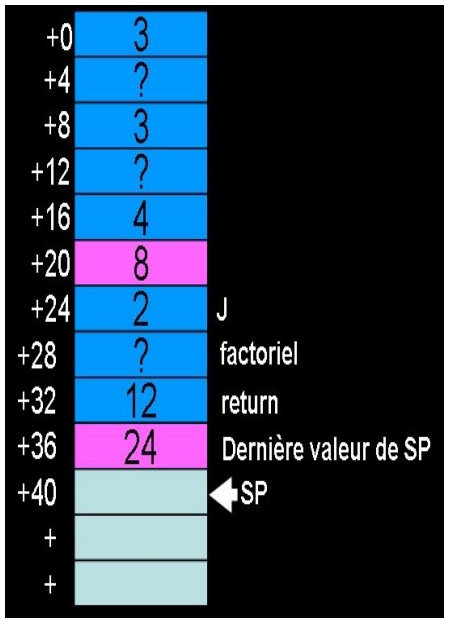
Les schémas suivants illustrent les différentes étapes d'empilement et de dépilement qui correspondent à l'exécution de ce programme pour une valeur de 3.

Appel factoriel(3)

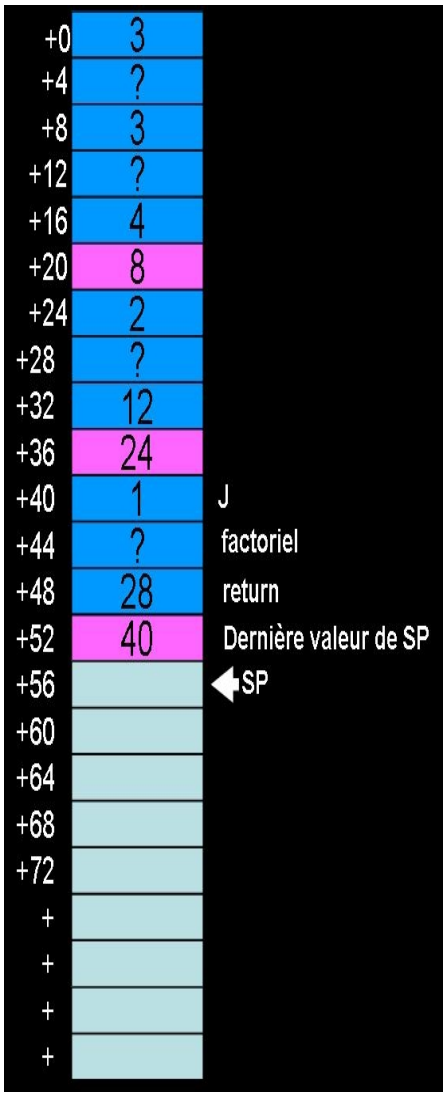
Appel factoriel(2)



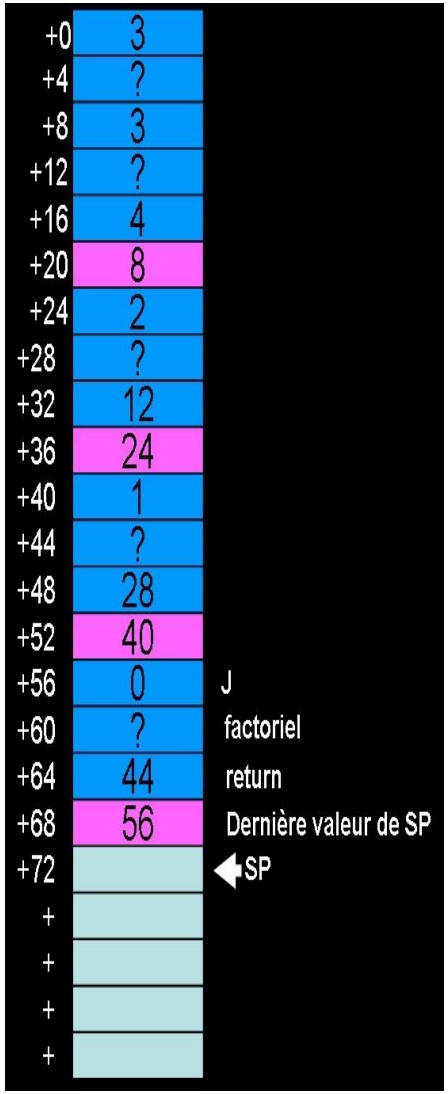
Appel factoriel(1)



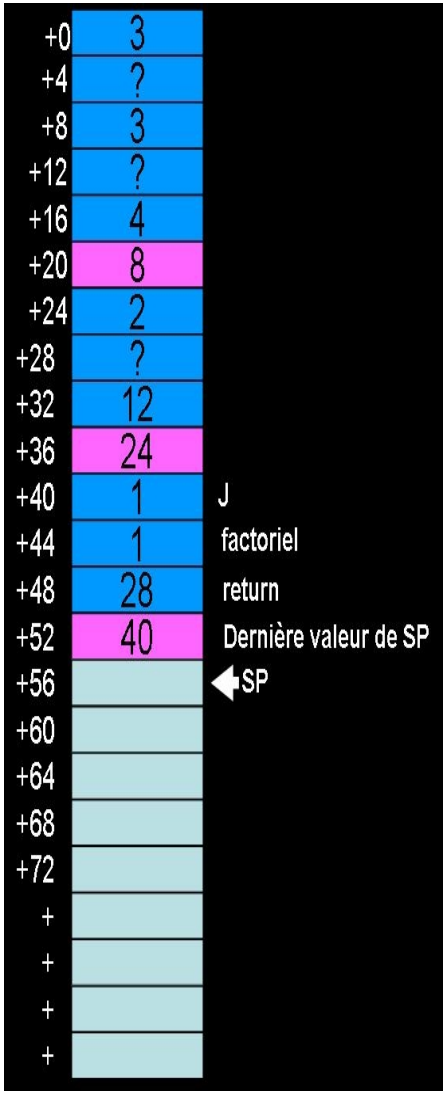
Appel factoriel(0)



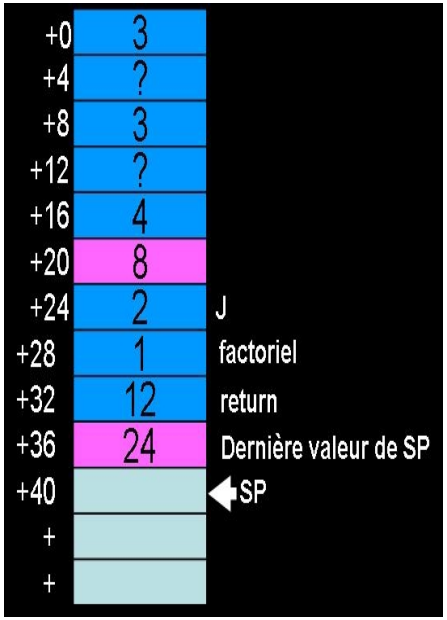
Affectation de 1 dans factoriel (0)



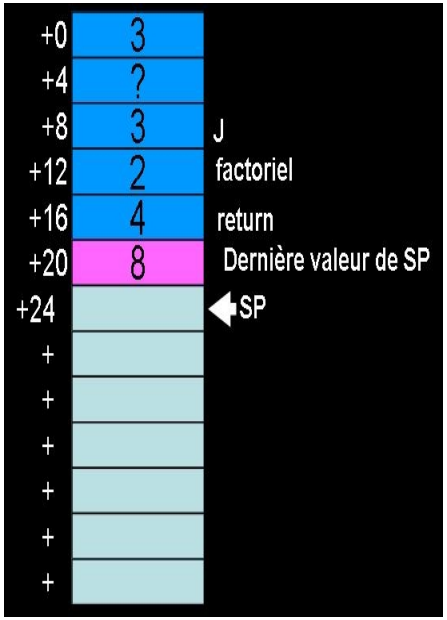
Retour dans factoriel(1)



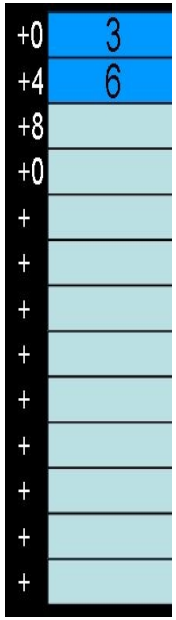
Retour dans factoriel(2)



Retour dans factoriel(3)



Retour (f




```
#include <stdio.h>
/* version de l'appel de factoriel en mettant le résultat en argument */
```

```
/* la valeur de retour de la fonction est toujours 0, car il n'y a pas de conditions d'erreur*/

int main()

{
int retour,i,resmain;


int factoriel(int j,int *resultat)
{int resfonc,retour;

if (j== 0) {*resultat=1;return 0;}/* critère d'arrêt de l'appel récursif. Rangement de la valeur 1 à
l'adresse passée par la fonction appelante*/

*/

retour = factoriel(j-1,&resfonc);/* appel récursif avec la valeur décrémentée de 1*/
*resultat=resfonc*j;/* multiplication par la valeur courante et rangement du résultat à l'adresse
passée par la fonction appelante*/
return 0;

};

do
{
printf("taper un entier (entre 0 et 16) :");
scanf("%i",&i);
}
while ((i<0) || (i>16));/* on boucle sur la saisie tant que i < 0 ou i>16*/

retour=factoriel(i,&resmain);

printf(" %i ! = %i \n",i,resmain);

return 0;
```

```
}
```