Cours informatique : Maîtrise de Sciences Cognitives 2002-2003

Session 2 :

Structure d'un programme

Rappel sur la notion de processus

Gestion de l'allocation mémoire pour les variables

- **Variables de type simple et tableau à une dimension**
- **Tableau à 2 dimensions**
- **Chaîne de caractères**

Structure d'un programme

Un programme source en C est constitué

- de directives de précompilations (lignes commençant par #)
- de commentaires (limités par /* et */)
- des déclarations ou définitions globales de variables ou de fonctions (vues de tout le programme)
- une fonction principale appelée "main" qui est le point d'entrée principal dans le programme au moment de son exécution
- un bloc principal lié à main délimité par { et }
- et à l'intérieur du bloc principal
  - des déclarations ou définitions de variables ou fonctions locales au programme principal (mais globales pour les fonctions incluses!)
  - des instructions ou des blocs d'instructions délimités par { et }
- toutes les instructions ou déclarations ou définitions sont suivies d'un ";"

Exemple sur le dernier exercice :

#include<stdio.h> /*Directive de précompilation (le préprocesseur va inclure le fichier entête stdio qui "déclare" les fonctions standard d'entrée-sortie*/

/* pas de déclaration ni définition globale*/

```
int main() /* définition de la fonction principale main qui dans cet exemple est supposée retourner un entier et ne possède pas d'arguments d'appel*/

{/* accolade de début de la fontion principale */

char c; /* déclaration d'une variable de type "char"*/

printf("Entrez un caractère : \n (p pour quitter) \n"); /* appel de la fonction printf avec un seul argument : une constante chaîne de caractères*/

/* la valeur retournée par la fonction printf n'est pas exploitée ici */

for (c=` `;c!=`p`; ) /* Pour (c étant initialement avec la valeur ` ` (espace);jusqu'à ce que c soit différent de la valeur `p`; pas d'opération particulière à chaque itération*/

{/* accolade de début du bloc d'instructions du "for"*/

c=getc(stdin); /* appel de la fonction "getc" avec comme argument "stdin"; la valeur retournée est mise dans la variable c*/

putc(c,stdout); /*appel de la fonction "putc" avec comme arguments "c" et "stdout"; la valeur retournée n'est pas exploitée*/

}/* accolade de fin du bloc d'instructions du "for" */

printf(" /n au revoir /n");

return 0;/* positionnement de la valeur 0 à retourner par la fonction principale ; cette valeur pourra être testée par le programme (le plus souvent l'interpréteur de commande qui a lancé l'exécution de ce programme*/

}/* accolade de fin de fonction principale */
```

Rappel sur la notion de processus

Un processus est un "programme en train de s'exécuter". C'est le système d'exploitation qui contrôle cette exécution. Chaque processus est identifié par le système d'exploitation par un Identificateur de Processus (Pid). Chaque processus est caractérisé par les zones mémoires qu'il occupe pour ses données, son code et son descripteur, par des droits, par des flux d'entrée-sortie ouverts pour lui. Par défaut, tout processus se voit attribuer 3 flux d'entrée-sortie standard (stdin, stdout et stderr).

Il faut voir les entrées-sorties standard comme des "flux" d'octets qui sont disponibles en entrée ou en sortie du processus. En entrée ces flux sont alimentés par un périphérique ou un fichier ou un flux de sortie d'un autre processus. En sortie, le flux est traité par un périphérique ou dirigé vers un fichier ou vers un flux d'entrée d'un autre processus (voir illustration en figure 2)

Un programme en train de s'exécuter utilise essentiellement 2 segments mémoires : un segment de mémoire pour les données et un segment de mémoire pour le code exécutable.

Toute référence à une donnée est faite de manière relative au début du segment de donnée.

Le segment de données alloue la mémoire selon deux techniques : la "pile" et le "tas". Le segment de code contient les instructions qui agissent sur les données exprimées en tant que adresse relative dans le segment de données.

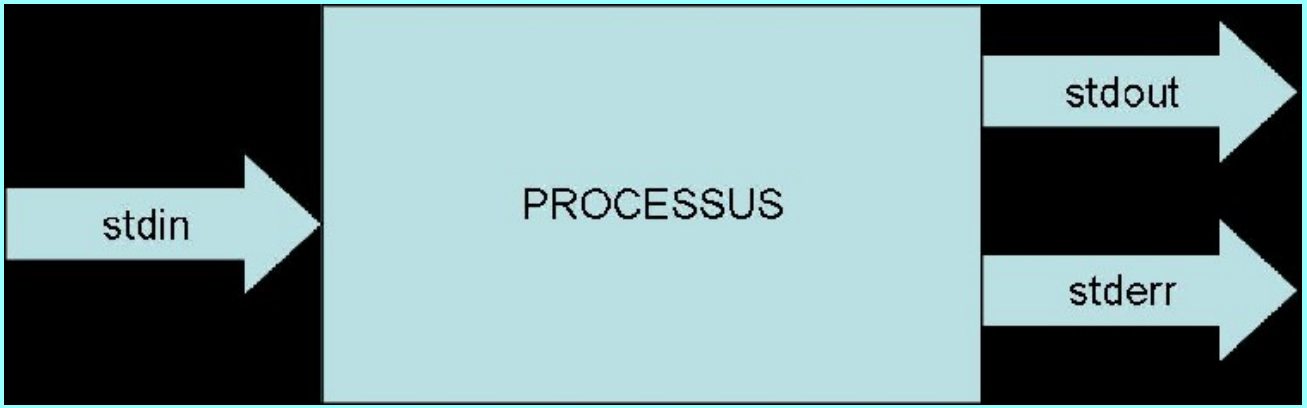


Figure 1 : les flux d'entrée-sortie standard d'un processus

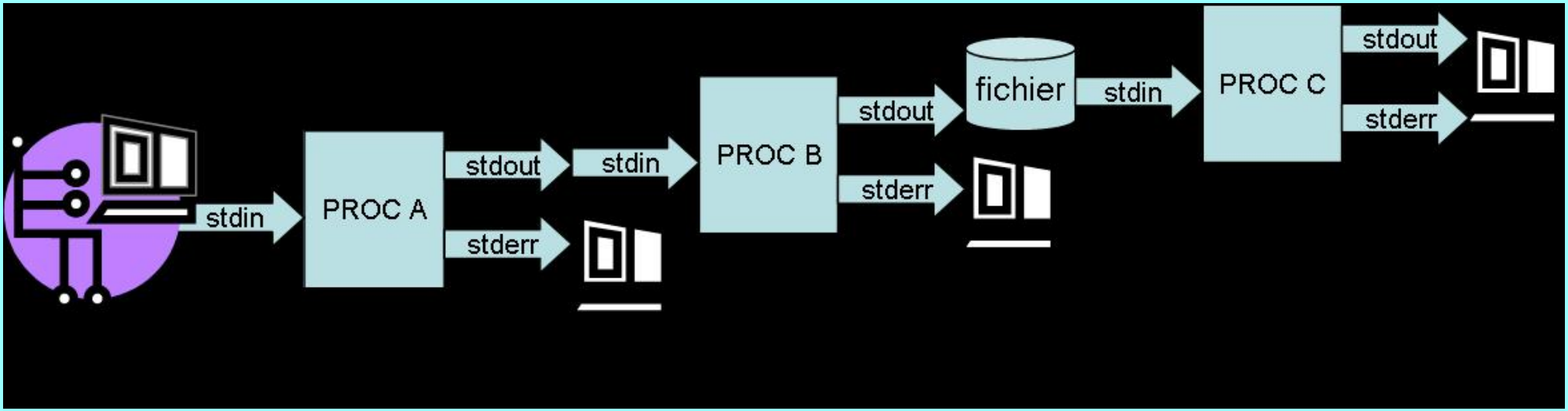


Figure 2 : exemple d'enchaînement de flux entre processus (notion de redirection de flux)

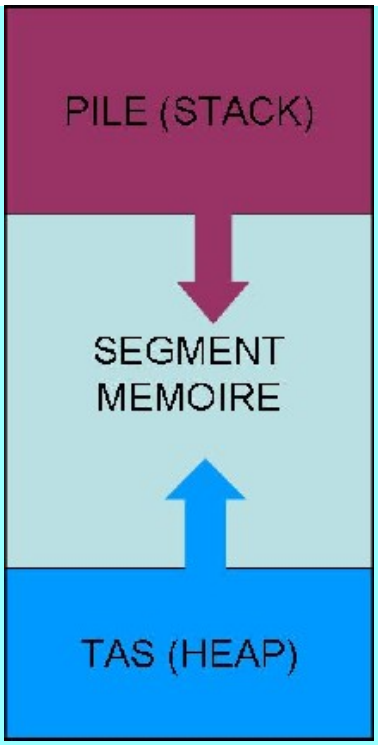


Figure 3 : l'allocation d'un segment de mémoire ; notion de pile et de tas.

Gestion de l'espace mémoire pour les variables

Nous allons regarder quelques exemples de programmes pour illustrer comment les variables déclarées sont gérées en mémoire.

Pour la syntaxe de déclaration des variables simples, cliquez ici.

Pour la syntaxe de déclaration des tableaux (et des structures!), cliquez ici.

EXEMPLE 1 (variables de type de base et tableau à une dimension):

```
#include <stdio.h>

int main()

{
int i; /* déclaration d'un entier i */
char car; /* déclaration d'un caractère car */
char cartab[9]; /* déclaration d'un tableau de 10 caractères */


car='a'; /* initialisation de car avec le code du caractère 'a' */
for (i=0;i<10;i++) /* i++ incrémente de 1 la valeur contenue dans i */
```

```
{cartab[i]=car;
car=car+1; /* incrément du code du caractère contenu dans car d'une valeur de 1 */
};
for (i=9;i>=0;i--) /* j-- décrémente de 1 la valeur contenue dans j */
printf("cartab[ %i ] = %c \n",i,cartab[i]);

return 0;
}
```

Cet exemple manipule plusieurs variables dont une variable structurée -un tableau- qui vont avoir un espace réservé dans le segment de données comme illustré dans la figure suivante :

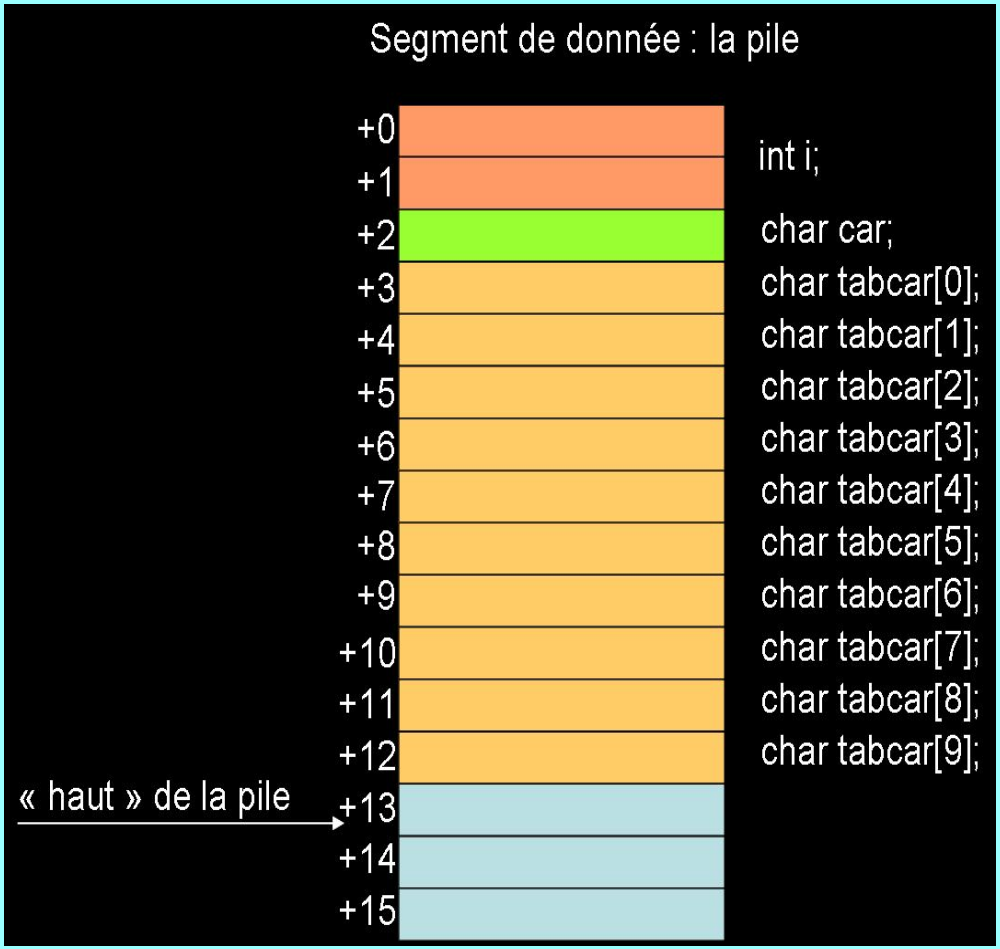


Figure 4 : Allocation mémoire des variables de l'exemple 1

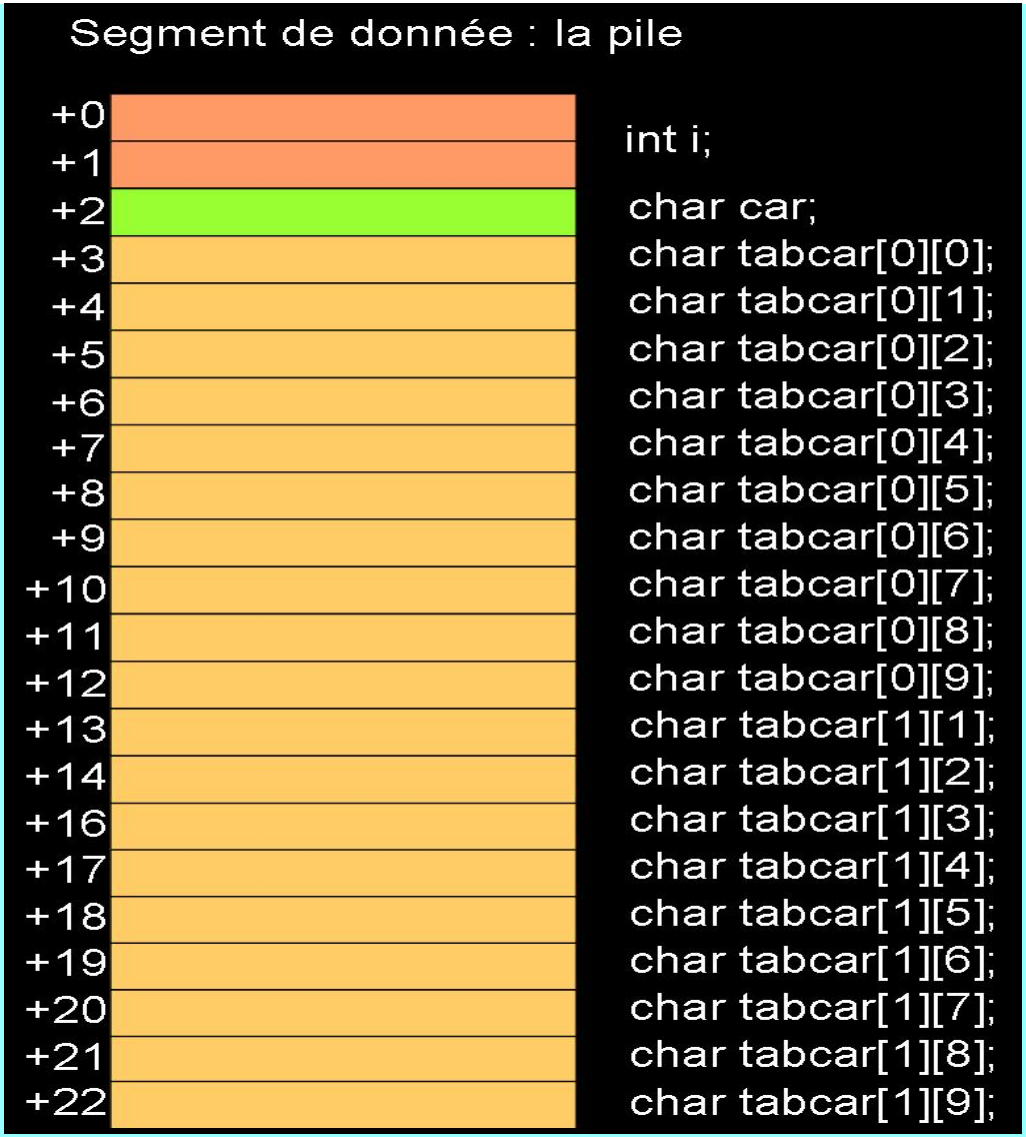
Comme on le voit, les caractères et les entiers ne prennent pas tous la même place et du coup ne peuvent pas représenter le même nombre de valeurs différentes.

Pour avoir une idée de quelques tailles occupées par les variables de type simple, cliquez ici.

EXEMPLE 2 : gestion d'un tableau à deux dimensions.

```
#include <stdio.h>

int main()

{
int i,j;
char car;
char cartab[9][9];

car='a';
for (i=0;i<10;i++)


{
for(j=0;j<10;j++)
{
cartab[i][j]=car;
car=car+1;
};
};
for (i=9;i>=0;i--)
{
for(j=9;j>=0;j--)
printf("cartab[ %i ][ %i ] = %c \n",i,j,cartab[i][j]);
};
}
```

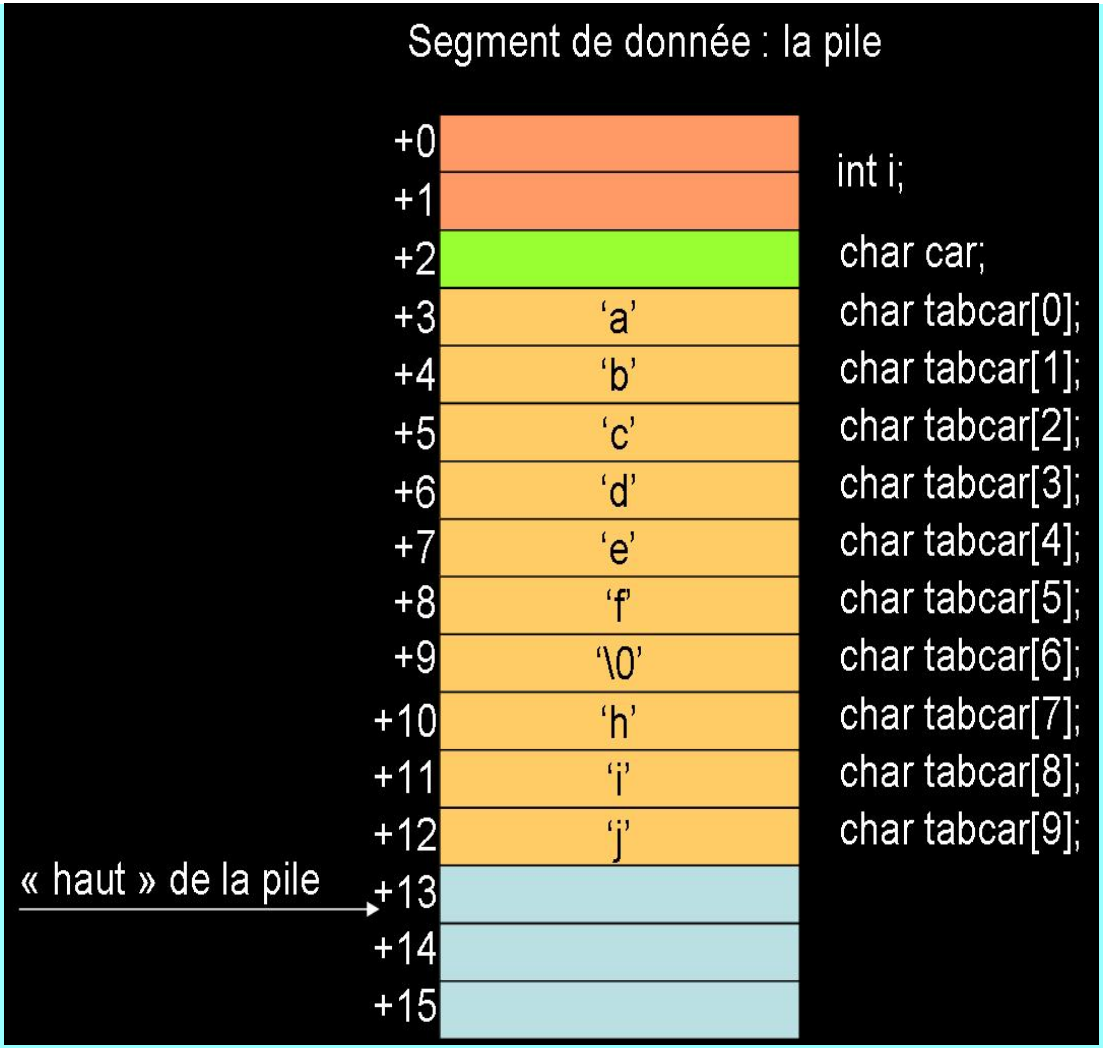
La figure suivante illustre l'état de l'allocation mémoire pour cet exemple.



Chaîne de caractères (pour la syntaxe et d'autres explications, essayez ici ou ici)

Il n'existe pas de "type" chaîne de caractères en langage C. Pour manipuler des chaînes, on utilise des tableaux de caractères avec une convention : le dernier caractère de la chaîne est le caractère '\0' (le caractère nul). Un ensemble de fonctions de manipulations de chaînes de caractères sont disponibles pour comparer, écrire, lire, copier, etc. Beaucoup de ces fonctions sont "dangereuses" : en effet, si elles ne rencontrent pas le caractère '/0' pour s'arrêter, elles peuvent largement déborder l'espace mémoire réservé pour la structure et produire des effets "imprévisibles" (en général, le programme se "plante" !).

La figure suivante illustre le cas d'une chaîne de caractère "abcdef" stockée dans un tableau de 10 cases.



Le programme suivant remplit un tableau avec 10 caractères, positionne le caractère nul en 7ème position, et une fonction d'écriture sur le flux de sortie standard imprime la chaîne de caractères ainsi constituée.

```
#include <stdio.h>

int main()

{
int i;
char car;
char cartab[9];


car='a';
for (i=0;i<10;i++)

{cartab[i]=car;
car=car+1;
```

```
};
for (i=0;i<10;i++)
printf("cartab[ %i ] = %c \n",i,cartab[i]);

cartab[6]= '\0';
printf(" et maintenant la chaine de caracteres constituee est : ");
i=puts(cartab);

return 0;
}
```