

Syntaxe du C: pointeurs, tableaux, structures

Aide-mémoire

MP2I Lycée Pierre de Fermat

Pointeurs

Déclaration, référencement

```
float x = 2.0;
float *p = &x; //p contient l'adresse de x
```

Déréférencement Lorsqu'une fonction doit déréférencer un pointeur, on s'assurera d'avoir mis une assertion du style `assert(p != NULL)`;

```
float y = *p; // y contient la valeur pointée par p, c'est à dire 2.0

*p = 9; // change la case pointée par p, c'est à dire x, en 9.
```

Affichage %p

```
float x = 2.0;
float *ptr = &x; //ptr contient l'adresse de x

printf("%p\n", ptr); // affiche l'adresse mémoire de x
```

Arithmétique des pointeurs

```
char* p = 0;
p = p + 1; // p contient 1 car char fait 1 octet

int* p2 = 0;
p2 = p2 + 1; // p2 contient 4 car int fait 4 octets
```

Tableaux statiques

Déclaration, définition

```
// crée un tableau de 23 entiers. Les valeurs initiales
// sont indéfinies
int tab1[23];

// crée un tableau de 6 flottants, de valeurs initialisées:
float tab2[6] = {0.1, 0.3, -98, 6.3, -6.3, 1};
```

Accès, modification

```
float tab[6] = {0.1, 0.3, -98, 6.3, -6.3, 1};

float x = tab[0]; // assigne 0.1 à x
tab[2] = 2.5; // remplace -98 par 2.5 dans tab

// écris 0 dans chaque case du tableau tab
for (int i=0; i<6; i++){
    tab[i] = 0;
}
```

Affichage Rappel : un tableau est un pointeur :

```
// crée un tableau de 6 flottants, de valeurs initialisées:
float tab[6] = {0.1, 0.3, -98, 6.3, -6.3, 1};

//affiche l'adresse du premier élément de tab
printf("%p\n", tab);

//affiche le premier élément de tab
printf("%f\n", tab[0]);
```

Chaînes de caractères

Opérations de base Une chaîne de caractère est un tableau de `char`, c'est à dire un pointeur vers un ou plusieurs `char`. Le type d'un chaîne de caractères est donc `char*`. En anglais on dit "character string", d'où le mot "string", souvent abrégé en "str".

Tout string se termine par le caractère nul, qui vaut 0, c'est à dire 0x00. On le note aussi `'\0'`.

```
// str1 doit avoir au moins 8 cases, car
// 7 lettres + 1 caractère nul.
char str1[8] = "Bonjour";
printf("%d\n", str1[1]); // affiche 66
printf("%d\n", str1[7]); // affiche 0

// Le format "%c" permet d'afficher un caractère et pas sa
// valeur numérique
printf("%c\n", str1[1]); // affiche 66

// On peut stocker une petite chaîne dans
// un tableau plus grand
char str[100] = "Coucou";
```

Affichage "%s"

```
char bjr[20] = "Bonjour les MP2I";
printf("%s\n", bjr); // affiche "Bonjour les MP2I"

// si l'on insère un caractère nul au milieu du mot
// alors on réduit effectivement la chaîne de caractères:
bjr[7] = 0;
printf("%s\n", bjr); // affiche juste "Bonjour"
```

Allocation dynamique

Allocation Il faut inclure la librairie <stdlib.h>

```
// réserve 62*4 = 248 octets de mémoire, et renvoie
// un pointeur vers le premier
int* p = malloc(62*sizeof(int));
```

Désallocation TRES IMPORTANT pour ne pas avoir de fuite mémoire

```
// réserve 62*4 = 248 octets de mémoire, et renvoie
// un pointeur vers le premier
int* p = malloc(62*sizeof(int));

// libère la mémoire allouée pour p
free(p);
```

Attention, si vous assignez à p deux valeurs, la première est perdue et ne peut pas être free :

```
// réserve 62*4 = 248 octets de mémoire, et renvoie
// un pointeur vers le premier
int* p = malloc(62*sizeof(int));

p = malloc(2*sizeof(int));

// libère les deux cases int allouées pour p en dernier
free(p);

// Les 62 premières cases réservées sont perdues à jamais
// dans le néant
```

Structures

Définition d'un type struct Pour définir un type représentant des humains ayant un nom, un prénom, un âge et une taille en mètres :

```
struct humain {
    char nom[20];
    char prenom[20];
    unsigned int age;
    float taille;
};
```

Déclaration d'une variable de type struct, accès aux attributs

```
struct humain x; // variable aux attributs non initialisés
x.nom = "Turing";
x.prenom = "Alan";
x.age = 110;
x.taille = 1.78;
```

Déclaration d'une variable de type struct initialisée

```
struct humain x = {  
    .nom = "Turing",  
    .prenom = "Alan",  
    .age = 110,  
    .taille = 1.78,  
}; // variable initialisée directement
```

Alias de type

```
typedef int my_type;  
my_type x = 8;
```

On peut combiner les alias de type avec les structures :

```
typedef struct humain {  
    char nom[20];  
    char prenom[20];  
    unsigned int age;  
    float taille;  
} humain_t;  
  
humain_t x; // pareil que struct humain x;
```

Pointeurs de structures

```
struct humain x = {  
    .nom = "Turing",  
    .prenom = "Alan",  
    .age = 110,  
    .taille = 1.78,  
};  
  
humain_t* ptr = &x;  
  
// Les deux syntaxes suivantes sont équivalentes :  
(*ptr).prenom;  
ptr->prenom; // on utilise plutôt celle là pour + de clarté
```