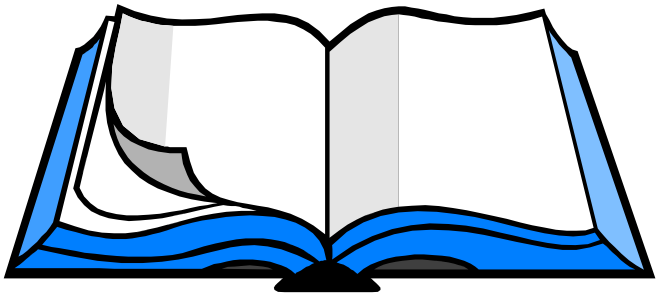
IUT de Lannion
Département Informatique
Semestre 3

Ludovic Liétard

R3.02 Développement efficace

Support de cours
(Les Pointeurs)



1. Les Pointeurs

1.1 Introduction

1.2 Déclaration de type et variables

1.3 Initialisations

1.4 Accès

1.5 Pointeur sur structure

1.6 Procédure et fonction

1.7 Exemples d'utilisation

1.1 Introduction

Pointeur est un type prédéfini.

Une variable de type pointeur contient l'adresse (la référence) d'un objet de type particulier.

Egalité entre pointeurs : Egalité des adresses et non pas des objets.

On parlera de pointeur sur un type.

Exemples :

pointeurs sur entier ;

pointeurs sur réel ;

pointeurs sur un type structure ;

1.2 Déclaration de pointeur

```
<Type_pointé> *<Variable_pointeur>;
```

Exemple

```
int *ref;
```

ref est une variable qui contiendra l'adresse d'une variable de type entier (l'adresse d'un entier).

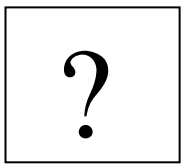
ref est une variable de type « pointeur sur entier ».

Attention

a) *ref* n'est pas de type entier.

b) *ref* contiendra l'adresse d'un entier.

c) *ref* ne contient rien après sa déclaration. On dit que référence n'est pas significatif et l'on note :



En d'autres termes, l'entier référencé n'existe pas au moment de la déclaration du pointeur.

d) La constante NULL indique que le pointeur ne pointe sur rien.

Après l'affectation :

ref= NULL;

nous avons :

ref

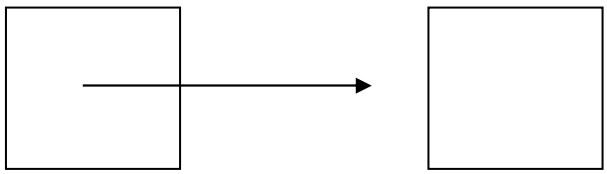
1.3 Initialisations

Un pointeur contient une adresse. Il est possible d'y mettre l'adresse d'une variable déjà existante (obtenue par &).

```
int *p;
int x;

main(){
        ........
        p=&x;
        ........
}
```

On note :



P X

Il est également possible de **créer une nouvelle variable en cours d'exécution** (allocation dynamique).

Il faut utiliser la primitive malloc :

Soit p un pointeur sur <type>. Il faut écrire :

p=(<type> *) malloc(sizeof(<type>));

remarque :

1) **sizeof(<type>)** est la taille prise par une donnée du <type> en mémoire; on réserve donc une fois cette taille.

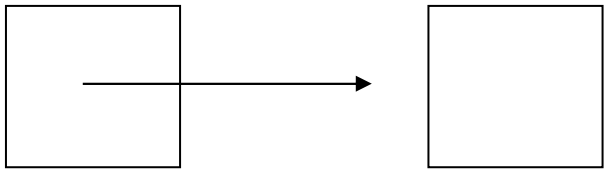
2) on convertit le résultat du malloc en pointeur sur type (parenthèse):

p=(**<type>** *) malloc(sizeof(<type>));

int *p;

```
int main(){
        ........
        p=(int *) malloc(sizeof(int));
        ........
    }
```

On note :



P

Cette deuxième façon est l'allocation dynamique.

1.4 Accès

*p est l’objet pointé par p

*p est une variable ordinaire accédée via son adresse

1.5 Pointeur sur structure

Soit un type structure :

```
typedef struct {
      type1 champ1 ;
      type2 champ2;
      etc..
} t_structure;
```

et un pointeur sur structure :

```
t_structure* pS;
```

On peut déclarer une structure par de l'allocation dynamique :

```
pS = (t_structure *)
malloc(sizeof(t_structure)) ;
```

et l'accès aux champs se fait par :

```
(*pS).champ1
```

que l'on écrira en C :

```
pS->champ1
```

qui est une variable de type1 : **c'est la rubrique champ1 de la structure pointée par pS.**

Cas particulier très important pour la suite

```
typedef struct Elem{
   int val ;
   struct Elem * svt;
} element;

typedef element* Liste;
```

1.6 Procédure et fonction

En langage C il n'y a que des fonctions. Une procédure est une fonction qui ne retourne rien (le type vide **void**).

Procédure en langage C et paramètre E/S :

```
void ajout(int *n, int a){
   *n=*n+a;
}
```

à l'appel :

```
   ajout(&x,7);
```

Fonction en langage C :

```
int ajout(int n, int a){
   return(n+a);
}
```

à l'appel :

```
   y = ajout(x,7);
```

1.7 Exemples

Exemple 1:

La saisie d'une variable au clavier se fait en utilisant la primitive *scanf*. Comme on modifie la variable, *scanf* à besoin de son adresse :

```
int a;
float f;

scanf("%d",&a);
scanf("%f",&f);
```

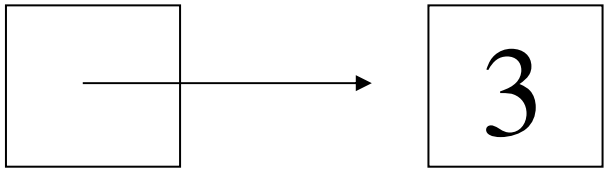
On utilise le même format que pour *printf*.

Exemple 2 :

```
int *p;
int x;
```

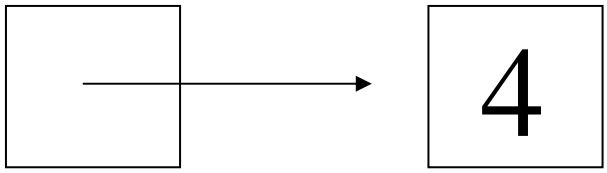
On peut effectuer le code suivant :

```
x=3;
p=&x;
```



P X

```
*p = *p + 1;
```



P X

Exemple 3 :

```
int *p;

p=(int *) malloc(sizeof(int));
*p = 9;
printf("%d",*p);
scanf("%d",p);
*p=*p+1;
```

Exemple 4 :

Le type tableau en C est une facilité accordée au programmeur.

a) Le nom d'un tableau de taille n est un pointeur sur une suite contigue de n variables.

```
int tab[10]
```

signifie :

tab

et l'on aurait pu écrire :

```
int * tab;
tab = (int *)  malloc(10*sizeof(int))
```

b) en conséquence :

tab[i] est équivalent à *(tab+i)

ce qui explique que les indices doivent obligatoirement commencer à 0 car le premier élément est à l'adresse tab .

Exemple 5 :

Les chaînes de caractères sont des tableaux de caractères qui se terminent par le caractère '\0' (la fin de chaîne).

On manipule les chaînes de caractères par les primitives de la bibliothèque <string.h>.