Support de cours
Les pointeurs.

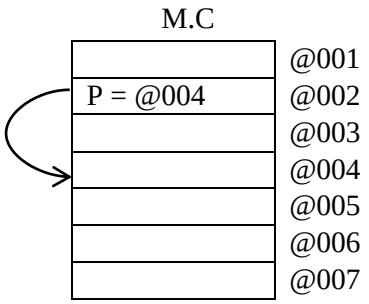
Introduction

Les cellules de la mémoire centrale (RAM) sont regroupées en octet et chaque octet est numéroté par un numéro unique appelé adresse mémoire. Déclarer une variable dans programme consiste à donner un identificateur à une zone de la MC. Pour manipuler cette zone mémoire il suffit d'utiliser son identificateur. Ce genre de variables s'appelle `'variables statiques'`, l'espace mémoire nécessaire est réservé dès l'exécution du programme. Les variables statiques ont l'inconvénient du gaspillage et parfois d'insuffisance de la mémoire. La solution c'est les variables dynamiques qui peuvent être réalisée grâce au type pointeur.

Définition

Le type pointeur est destiné pour contenir une adresse mémoire. Une variable de type pointeur est stockée dans une case mémoire et contient l'adresse d'une autre case mémoire.

Exemple : Si on considère un pointeur P contenant l'adresse mémoire @004, la situation en mémoire centrale est représentée ci-dessous. On dit que P pointe la case mémoire à l'adresse @004.



Déclaration

Pour déclarer une variable de type pointeur, il faut préciser le type de la case mémoire à pointer. La syntaxe pour déclarer un pointeur est la suivante

syntaxe	
Algorithmique	**Langage C**
`Var id_pointeur : ↑type;`	`type *id_pointeur;`
Exemples	
`Var P : ↑entier;`	`int *P;`

`Type pointeur = ↑Etudiant;`
`Etudiant = enregistrement`
`Nom: chaine[30];`
`Prenom : chaine[30];`
`Moyenne : reel;`
`Var q : pointeur;`
	`typedef struct Etudiant Etudiant;`
	`Etudiant *q ;`
	`struct Etudiant{`
	`char Nom[30];`
	`char Prenom[30];`
	`float Moyenne ;`
	`};`

Dans l'exemple ci-dessus, P est un pointeur destiné à contenir une adresse mémoire d'une case de type entier et **q** est destiné à contenir l'adresse d'une variable de type `Etudiant`. Il est possible de déclarer un pointeur vers un type qui n'est pas encore défini, car la taille d'une case mémoire de type pointeur est la même quel que soit le type pointé (un pointeur est codé généralement sur 4 octets).

Support de cours
Les pointeurs.

Manipulation des pointeurs en algorithmique

Après déclaration d'une variable de type pointeur, celle-ci ne contient aucune adresse et elle ne pointe vers aucune case mémoire. Sa valeur est la valeur par défaut '**NIL**' et signifie que le pointeur ne pointe nul part. Pour réserver la case mémoire d'un pointeur il faut utiliser la procédure prédéfinie `Allouer(Ptr)` qui réserve une case mémoire et stocke sont adresse dans le pointeur `ptr`. Pour accéder et manipuler le contenue de la case mémoire pointée par `ptr` en écrit `ptr↑`

Exemples :

```
Var p : ↑entier;
allouer(p);
p↑ ← 25 ;
```

L'instruction `allouer(p);` permet de réserver un espace mémoire pour un entier et son adresse est stockée dans le pointeur p. p↑ désigne la case mémoire pointé par p.

```
Algorithme pointeur ;
Type Etudiant = enregistrement
       Nom: chaine[30];
       Prenom : chaine[30];
       Moyenne : reel;
Var q : ↑Etudiant;
Debut
|    Allouer(q);
|    Avec (q↑) faire
|    |  Lire(nom,prenom,moyenne);
|    finAvec;
|     ecrire(q↑.nom, q↑.prenom);
Fin.
```

Les variables de type pointeur sont appelées '`variables dynamiques`' l'espace mémoire nécessaire est alloué dans la partie action de l'algorithme. L'espace mémoire réservé avec la procédure `Allouer` reste occupé même après terminaison du programme. Dans un algorithme il faut libérer tous les objets pointeurs en utilisant la procédure `liberer(ptr)` ; pour ne pas saturer la mémoire de la machine.

Remarque : la valeur 'NIL' d'un pointeur est très importante. Le test Si (ptr = NIL) permet de savoir si le pointeur contient une adresse.

Manipulation des pointeurs en Langage C

L'opérateur unaire d'indirection * permet d'accéder directement à la valeur de l'objet pointé. Ainsi, si p est un pointeur vers un entier i, *p désigne la valeur de i.

```
main()
{ int i = 3;
  int *p;
  p = &i;
  printf("*p = %d \n",*p);
}
```

Support de cours
Les pointeurs.

L'opérateur & renvoie l'adresse mémoire d'une variable, le pointeur p contient l'adresse de i. Alors, le programme affiche `*p = 3`.

Les opérations possibles sur un pointeur sont :

- L'addition d'un entier à un pointeur.
- La soustraction d'un entier d'un pointeur

Le type des pointeurs est important pour leur arithmétique. Si `i` est un entier et `p` est un pointeur sur un objet de type *`type`*, l'expression `p + i` désigne un pointeur sur un objet de type `type` dont la valeur est égale à la valeur de `p` incrémentée de `i * sizeof(type)`. Il en va de même pour la soustraction d'un entier à un pointeur.

Exemples :

```
main()
{
  int i = 3;
  int *p1, *p2;
  p1 = &i;
  p2 = p1 + 1;
  printf("p1 = %ld \t p2 = %ld\n",p1,p2);
}
```

Le programme affiche `p1 = 4831835984  p2 = 4831835988.`

Par contre, le même programme avec des pointeurs sur des objets de type `double` :

```
main()
{
  double i = 3;
  double *p1, *p2;
  p1 = &i;
  p2 = p1 + 1;
  printf("p1 = %ld \t p2 = %ld\n",p1,p2);
}
```

Affiche `p1 = 4831835984  p2 = 4831835992.`

Allocation dynamique

Avant de manipuler un pointeur, et notamment de lui appliquer l'opérateur d'indirection *, il faut l'initialiser. Sinon, par défaut, la valeur du pointeur est égale à une constante symbolique notée **NULL** définie dans `<stdio.h>`. En général, cette constante vaut 0. Dans ce cas le pointeur ne contient aucune adresse ainsi que le test p == NULL permet de savoir si le pointeur p pointe vers un objet. Pour pouvoir utiliser la variable *p pointée par p, il faut lui réserver un espace mémoire. Cette opération s'appelle *allocation dynamique*, elle se fait en langage C avec la fonction `malloc` prédéfinie dans `<stdlib.h>`, sa syntaxe est :

```
Pointeur = malloc(nombre-octets) ;
```

Cette fonction retourne un pointeur de type `char*` pointant vers un objet de taille nombre-octets. Pour initialiser des pointeurs vers des objets qui ne sont pas de type char, il faut convertir le type de la sortie de la fonction malloc à l'aide d'un cast. L'argument nombre-octets est souvent donné à l'aide de la fonction `sizeof(type)` qui renvoie le nombre d'octets utilisés pour stocker un objet.

Ainsi, pour initialiser un pointeur vers un entier, on écrit :

Support de cours
Les pointeurs.

```
#include <stdlib.h>
int *p;
p = (int*)malloc(sizeof(int));
```

Enfin, lorsque l'on n'a plus besoin de l'espace-mémoire alloué dynamiquement (c'est-`a-dire quand on n'utilise plus le pointeur p), il faut libérer cette place en mémoire. Ceci se fait à l'aide de l'instruction `free` qui a pour syntaxe

```
free(nom-du-pointeur);
```

A toute instruction malloc doit être associée une instruction free.

Pointeurs et tableaux

Tout tableau en C est en fait un pointeur constant dant la déclaration est :

```
int tab[10];
```

`tab` est un pointeur constant (non modifiable) dont la valeur est l'adresse du premier élément du tableau. Autrement dit, tab a pour valeur &tab[0]. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.

```
int tab[5] = {1, 2, 6, 0, 7};
main(){
int i; int *p;
p = tab;
for (i = 0; i < 5; i++){
printf(" %d \n",*p);
p++;
}}
```

Grace à l'allocation dynamique, il est possible de créer des tableaux dont la taille est connue à l'exécution du programme.

```
#include <stdlib.h>
main(){
int n;
int *tab;
scanf("%d",&n);
...
tab = (int*)malloc(n * sizeof(int));
...
free(tab);
}
```