Langage procédural

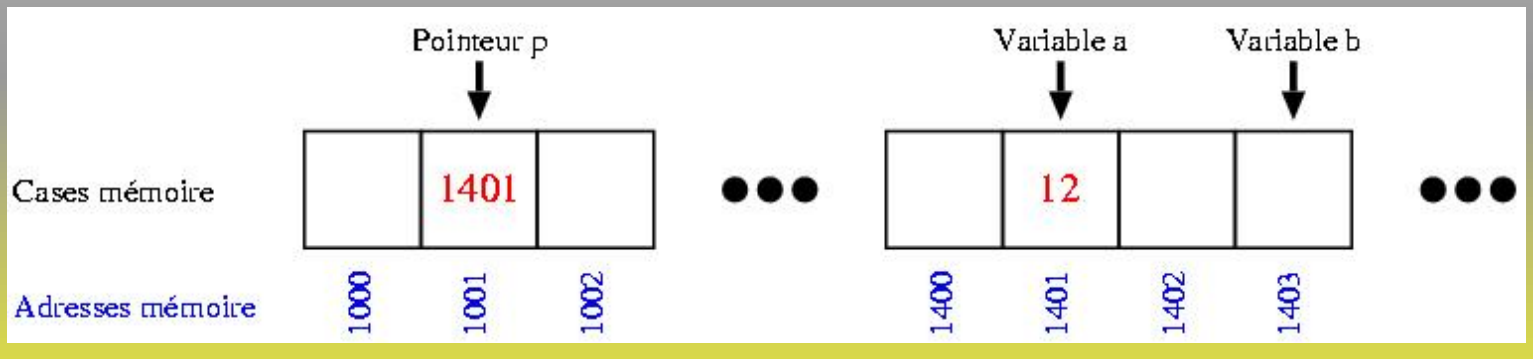
Le langage C

Nizar OUARTI

1 Les pointeurs

Adresse d'une variable de type simple

- int x ; On a vu que pour envoyer son adresse il faut écrire &x
- On peut créer une variable appelée pointeur qui stocke les adresses
- int *p
- Tel que p=&x ;//p stocke l'adresse de x
- et *p <==> x
- int y=*p <==> y=x



Moyens mnémotechniques

- int *p ; *p est un int
- int* p ; p est un int* (pointeur vers un int)
- Ces deux notations sont équivalentes
- Une variable c'est comme une boite au lettre, il y a l'adresse de la boite et son contenu (valeur) le courrier !
- **A retenir : Un pointeur stocke une adresse.**

Arithmétique

- *p à la même arithmétique que x
- int y= *p +1 ;// equivalent à int y= x+1 ;
- Important les parenthèses dans le cas (*p)++ ;
- Les importantes car ++ à une priorité plus forte et on aurait eu la valeur après incrémentation du pointeur

Arguments modifiables de fonction

- On a vu que pour scanf, le paramètre devait être une adresse pour être modifiable
- int a ; scanf("%d",&a) ;
- De même, int *pa ; scanf("%d",pa) ; est équivalent
- pa contiendra l'adresse de la variable et *pa contiendra la valeur de la variable
- Créer une fonction swap qui échange les valeurs de deux variables entières entrées en paramètres.

La fonction swap

- Swap

```
1  #include <stdio.h>
2
3  void  swap(int *px, int *py)
4  {
5     int temp=*px;
6     *px=*py;
7     *py=temp;
8  }
9  int main(int argc, char* argv[])
10 {
11    int x=2;
12    int y=5;
13    void  swap(int *px, int *py);// facultatif
14
15    swap(&x,&y);
16
17    return 0;
18 }
```

Pointeurs et tableaux

- int a[10] ;
- int *pa ; pa=&a[0] ; On crée un pointeur qui pointe vers le premier élément du tableau

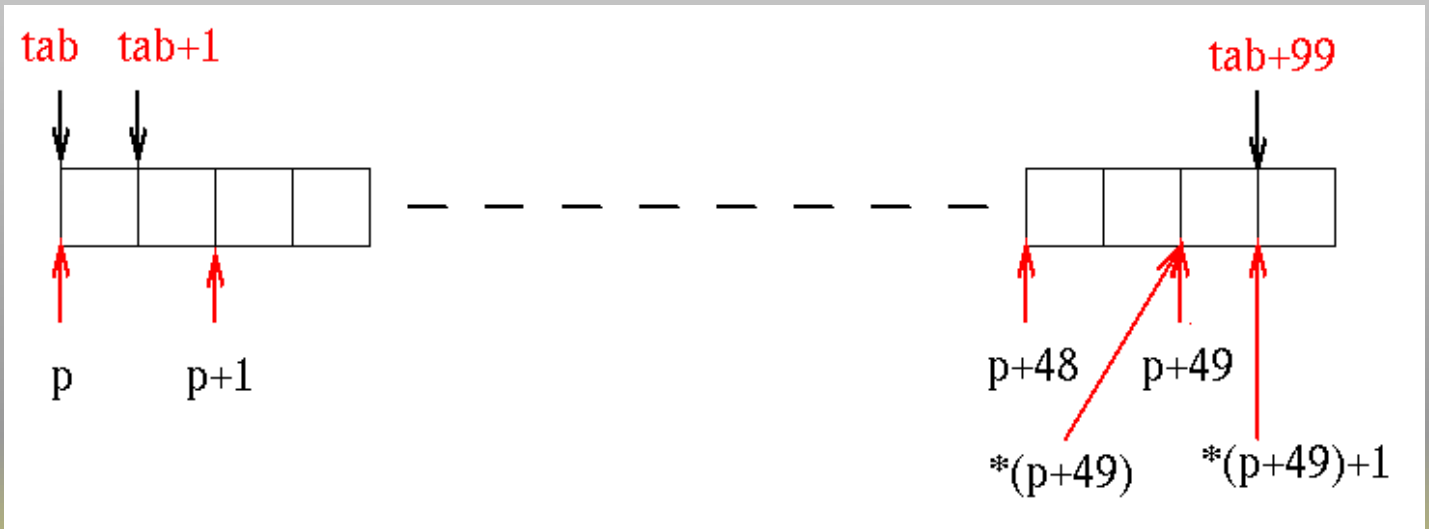


FIGURE: int tab[100] ; int *p ; p=&tab[0] ;

Arithmétique des pointeurs

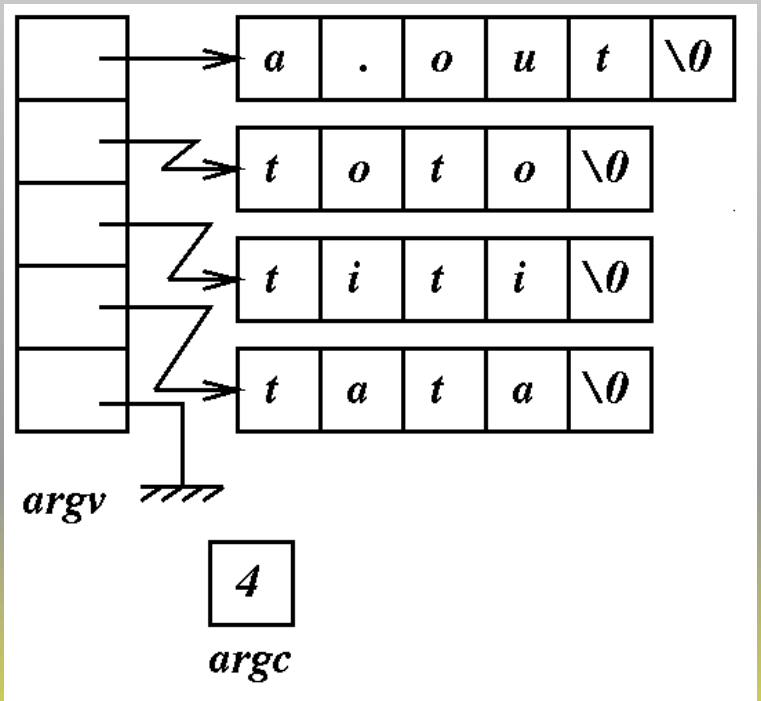
- Par contre maitenant on a la propriété suivante des pointeurs
- *(pa+1) <==> a[1] ; *(pa+n) <==> a[n]
- Lorsque l'on incrémente un pointeur, il fait un saut d'adresse de la taille du type vers lequel il pointe
- en réalité a est aussi un pointeur c'est le pointeur vers le premier élément
- a<==> &a[0]
- Donc on aurait pu écrire tout à l'heure pa=a ;
- On voit aussi qu'on peut manipuler un tableau avec *(a+n)
- Pour tout ceci attention de bien connaître la taille du tableau

Pointeurs vers tableau de char : string

- char *str ; est un pointeur vers un string
- c'est quasiment équivalent à char str[] ;

Pointeurs de pointeurs

- char **argv <==> char *argv[]
- Ceci est équivalent à des tableaux bidimensionels



Arguments par valeur ou par adresse d'une fonction

- Lorsque les arguments d'une fonction sont envoyés par adresse le contenu des variables en question est accessible et modifiable.
- Au lieu d'envoyer la valeur de la variable, on envoie son adresse
- L'opérateur pour faire ceci est **&** ou bien grâce au pointeur

Pointeurs vers une fonction

- Permet d'avoir des fonctions en arguments
- Il faut pour cela que le prototype possède un pointeur vers la fonction
- C'est à dire qu'il attende son adresse

```
1 /* Exemple fonction comme paramètre */
2 void func ( void (*f)(int) );// prototype
3 //il faut que l'argument soit bien du prototype voulu
4
5 void print ( int x ) {
6   printf("%d",x);
7 }
8
9 func(print);// usage
```

Pointeurs de fonction

- Un pointeur vers une fonction peut être appelé handle
- int (*f)() ici aucun paramètre précisé, forme très intéressante
- Cette forme accepte toute fonction ayant comme sortie int quel que soit le nombre ou le type de paramètres
- Quelle est la différence avec int *f(), les parenthèses étaient-elles obligatoires ?

Pointeurs et structures

- Les structures ne sont pas des pointeurs contrairement aux tableaux
- Donc si on veut modifier une structure dans une fonction, il faut envoyer son pointeur
- Point *p ; usage : (*p).x=0.5
- Parenthèses obligatoires ! ! !
- (*p).x <==> p− >x
- Vu que les pointeurs vers des structures sont très courants l'opérateur − > a été créé pour simplifier la notation

Exercice

- char *chaines[100] ;
- int mat[100][40] ;
- char **argv ;

Exercice

- char *chaines[100] ;
- int mat[100][40] ;
- char **argv ;
- un tableau de 100 pointeurs de caractère,

Exercice

- char *chaines[100] ;
- int mat[100][40] ;
- char **argv ;
- un tableau de 100 pointeurs de caractère,
- un tableau de 100 éléments, chaque élément étant un tableau de 40 entiers,

Exercice

- char *chaines[100] ;
- int mat[100][40] ;
- char **argv ;
- un tableau de 100 pointeurs de caractère,
- un tableau de 100 éléments, chaque élément étant un tableau de 40 entiers,
- un pointeur de pointeur de caractère.

Exercice

- int (*tab)[10] ;
- char (*f)() ;
- char *(*g)() ;
- float *(*tabf[20])() ;
- Trouver ce qui a été défini

Exercice

- int (*tab)[10] ;
- char (*f)() ;
- char *(*g)() ;
- float *(*tabf[20])() ;
- Trouver ce qui a été défini
- un pointeur de vecteur de 10 entiers,

Exercice

- int (*tab)[10] ;
- char (*f)() ;
- char *(*g)() ;
- float *(*tabf[20])() ;
- Trouver ce qui a été défini
- un pointeur de vecteur de 10 entiers,
- un pointeur de fonction retournant un caractère,

Exercice

- int (*tab)[10] ;
- char (*f)() ;
- char *(*g)() ;
- float *(*tabf[20])() ;
- Trouver ce qui a été défini
- un pointeur de vecteur de 10 entiers,
- un pointeur de fonction retournant un caractère,
- un pointeur de fonction retournant un pointeur de caractère,

Exercice

- int (*tab)[10] ;
- char (*f)() ;
- char *(*g)() ;
- float *(*tabf[20])() ;
- Trouver ce qui a été défini
- un pointeur de vecteur de 10 entiers,
- un pointeur de fonction retournant un caractère,
- un pointeur de fonction retournant un pointeur de caractère,
- un tableau de 20 pointeurs de fonction retournant un pointeur de réel.

Initialisation de pointeur

- Après avoir déclaré un pointeur il faut l'initialiser.
- Lorsque vous déclarez un pointeur, celui-ci contient ce que la case où il est stocké contenait avant, c'est-à-dire n'importe quel nombre.
- Si vous n'initialisez pas votre pointeur, celui-ci risque de pointer vers une zone hasardeuse de votre mémoire
- Cette zone peut être un morceau de votre programme ou... de votre système d'exploitation !

Tableaux dynamiques

- stdlib.h bibliothèque qui contient les function d'allocation de mémoire malloc, calloc realloc, free
- calloc et malloc pour réserver de la place en mémoire
- malloc : prototype : void * malloc (sizet t) avec t le nombre de bytes
- int * p ; p = (int*)malloc (sizeof(int)) ;
- important de tester (p==NULL) car c'est ce qui arrive quand l'allocation est impossible
- calloc : prototype :void * calloc (sizet n, sizet t) avec n le nombre d'éléments et t nombre de bytes
- calloc(n, t) <==> malloc(n * t) ;

Tableaux dynamiques

- realloc sert pour dynamiquement changer la taille d'un bloc mémoire sur lequel pointe le pointeur
- realloc : prototype : void * realloc (void * init , sizet t), avec init le pointeur initiale et t la taille en bytes à ajouter
- De la même manière on doit tester (p2==NULL) avec p2 = (int*)realloc (p,sizeof(int)) ;
- Très important, les données entrées dans p seront présentes dans p2
- Très important **free** sert à libérer la mémoire ! !
- free(p) ;

Astuce

- A la place de long *pl = (long*)malloc(sizeof(long)) ;
- Ecrire long *pl = (long*)malloc (sizeof (*pl)) ;
- identique, mais plus souple si on doit changer le type au moment de la programmation
- Dans le deuxième cas on ne change que 2 fois au lieu de 3