Programmation Impérative II

Tableaux et Pointeurs
Chaînes de Caractères

minh-anh.tran@u-pec.fr

Les Tableaux

Déclaration :

```
int Tab[20];
```

Eléménts du Tableau :

```
Tab[2] = 5; Tab[0] = 4;
printf ("%d\n ", Tab[2*i]);
scanf ("%d\n ", &Tab[5]);
```

Taille du Tableau :

une constante ou une expression constante

```
#define N 50
int T[N]; float X[2*N];
```

Les Tableaux

Aucun contrôle de débordement d'indice
dans la plupart des compilateurs

Initialisation:

```
int Tab[4] = {5,2,3,4};
Tab[0] = 5; Tab[3] = 4;
```

Transmission comme paramètre à une fonction :

taille fixe

```
void  fct (int t[10]);
{          }
```

Les Tableaux

taille variable

```
int somme (int t[], int nb)
{ int i, s=0;
   for (i=0; i<nb; i++)
      {s = s + t[i];}
   return (s);
}
int main (void)
{  int T[10];
   T[0] = 1; T[1] = 5; T[2] = 4; T[3] = 6;
   printf ("%d", somme(T,2));}
```

Les Pointeurs

Le langage C permet de manipuler des adresses par l'intermédiaire des variables de type **pointeur**

Déclaration

```
int * ad, n=4;
```

ad : une variable qui sera utilisée pour pointer sur des entiers
ad sera une adresse dont le contenu est un entier

Opérateurs & et *

```
ad = &n;
```

***adresse** d'une variable, déjà utilisé avec scanf*

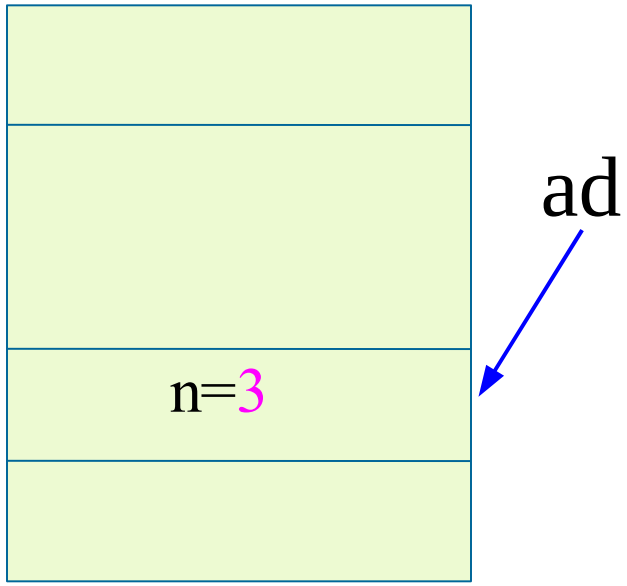
```
*ad = 4;
```

***contenu** d'un pointeur (à une adresse)*

Les Pointeurs

```
int  n;
int  *ad;
ad = &n;


*ad = 3;
//a le même effet que
n = 3;
```



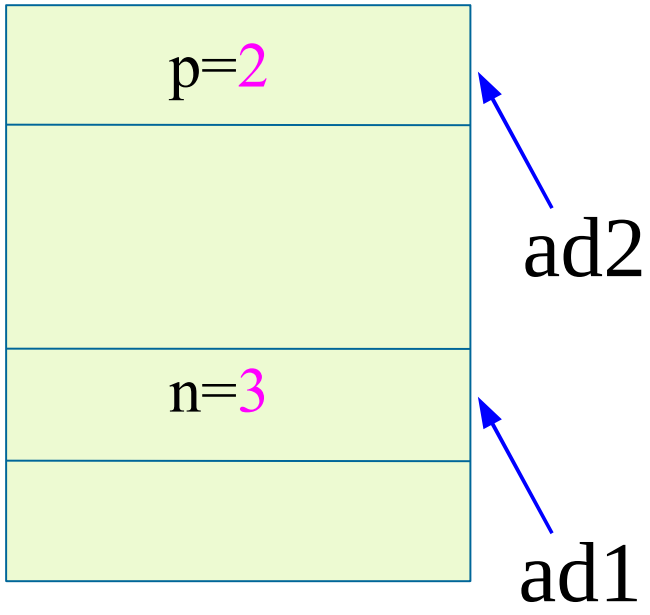
La variable de type pointeur est une adresse
qui peut pointer sur un type de donnée quelconque

Exemple

```
int * ad1, *ad2, n, p = 1;
char *ad3;

ad1 = &n; ad2 = &p;
*ad1 = *ad2 + 2;

printf ("%d %d \n", *ad1, *ad2);
```



Les Pointeurs

Rmq. *Le nom d'un tableau est un pointeur qui indique l'adresse du premier élément du tableau*

float T[10] T est un pointeur de type float

La notation T est équivalente à &T[0]

T+i	&T[i]
T[i]	* (T+i)

Les Pointeurs

Passage de paramètres

```
void  add1 (int a, int b, int c)
{  c = a + b;
        printf ("%d \n", c);}
void  add2 (int a, int b, int * c)
{  *c = a + b;
        printf ("%d \n", *c);}
int main (void)
{ int i = 1 , j = 2, k = 0;
        add1 (i, j, k);        printf ("%d \n", k);
        add2 (i, j, &k);     printf ("%d \n", k);
  return 0; }
```

Chaînes de Caractères

Tableau de caractères , le dernier élément étant le caractère ‘\0’

```
char * chaine;
char tab1[6]={‘a’,’b’,’\0’}, tab2[]="bonjour";
char tab3[5]="bonjour";
int main (void)
{
   tab1[0] = ‘a’;  tab2[3] = ‘x’;
   tab1 = "bonjour";
}
```

problème de la longueur

ce n’est pas possible

Lire et Ecrire des Chaînes

1-le format ***%s*** avec ***printf*** et ***scanf***
2-fonctions spécifiques ***gets*** et ***puts***

```
char s1[20], * s2;
gets (s1);
s2 = s1;
puts (s2);

scanf ("%s \n", s1);
s2 = s1;
printf ("%s \n", s2);
```

Fonctions prédefinies dans la bibliothèque

string.h

Soient ***n*** un entier,
et ***s*** et ***t*** deux chaînes de caractères

strlen(s) retourne la longueur de ***s*** sans compter '\0'
strcpy(s,t) copie ***t*** vers ***s***
strcat(s,t) ajoute ***t*** à la fin de ***s***
strncpy(s,t,n) copie ***n*** caractères de ***t*** vers ***s***
strncat(s,t,n) ajoute ***n*** caractères de ***t*** à la fin de ***s***

Ordre alphabétique des caractères

Dépend du code utilisé

Pour le code ASCII:

,0,1,2, ... ,9, . . . ,A,B,C, ... ,Z, . . . ,a,b,c, ... ,z, . . .

chaîne vide prècede toutes les autres chaînes

"ABC" précède "BCD" car 'A'<'B'

"ABC" précède "B" car 'A'<'B'

"Abc" précède "abc" car 'A'<'a'

"ab" précède "abcd" car "" précède "cd"

" ab" précède "ab" car ' '<'a'

strcmp(s, t) compare ***s*** et ***t*** lexicographiquement et fournit un résultat:

1. Négatif (-1) si s précède t
2. Zéro (0) si s est égal à t
3. Positif (1) si s suit t

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL