A3P(AL)

Cours C2

Sommaire

1. Adresse d'une variable
2. Valeur située à une adresse
3. Passages de paramètres
4. Équivalence pointeur ←→ tableau
5. Différences pointeur ←→ tableau
6. Pointeur de pointeur
7. Type pointeur

1. Adresse d'une variable

- Doit être contenue dans un « pointeur »
- Une variable peut contenir toutes les valeurs d'un certain type …
- `int vE = 12;` de type entier
- `int* vP;` de type pointeur d'entier

  `vP = &vE;` `vP` ← adresse de `vE`
- Un pointeur d'entier est différent d'un pointeur de réel ou de caractère
- ∃ des pointeurs génériques : `void *`

2. Valeur située à une adresse

- `printf( "%d", *vP );`
  `*vP = -5;`

- `int* vP;`
  `vP` est un pointeur d'entier

- `int *vP;`
  `*vP` est un entier

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- Que signifie **`*&vE`** ?

A **adresse de vE**	**D** **ne compile pas**
B **ce qu'il y a à l'adresse vE**	**C** **vE**
  ```

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- Que vaut **`&*vP`** ?

A **adresse de vP**	**D** **ne compile pas**
B **ce qu'il y a à l'adresse vP**	**C** **vP**
  ```

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- Que signifie **`*&vP`** ?

A **adresse de vP**	**D** **ne compile pas**
B **ce qu'il y a à l'adresse vP**	**C** **vP**
  ```

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- Que signifie **`&*vE`** ?

A **adresse de vE**	**D** **ne compile pas**
B **ce qu'il y a à l'adresse vE**	**C** **vE**
  ```

3. Passages de paramètres (1/2)

- En java, un seul mode :
  par valeur = par recopie
  => toujours paramètre d'entrée
  (pour fonctions et procédures)

- En C, deuxième mode :
  possible aussi par adresse
  => paramètre de sortie ou entrée/sortie
  (convention : uniquement pour procédures)

3. Passages de paramètres (2/2)

- Passage par adresse
  (paramètre de sortie ou entrée/sortie)
  => adresse d'une variable => pointeur

```
main: int vE=2; triple( &vE );
```

```
void triple( int* pE )
 { *pE = *pE * 3; }
```

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- ```
  scanf( "%d", ? );
  ```

A **vP**	**D** **autre**
B **&vP**	**C** ***vP**

QUIZZ

- ```
  int  vE = 12;
  int* vP = &vE;
  *vP = -5;
  ```

- ```
  printf( "%d", ? );
  ```

A **vP**	**D** **autre**
B **&vP**	**C** ***vP**

QUIZZ

- `char vTab[] = "Bonjour";`
- Le compilateur crée un tableau de combien de cases ?
- On peut écrire `printf( "%s", vTab );`

A **7 et oui**	**D** **8 et oui**
B **7 et non**	**C** **8 et non**

4.1 Équivalence pointeur ←→ tableau

- `char vTab[] = "Bonjour";`

- Si `&vTab[0]` vaut 800, alors
  `&vTab[1]` vaut 801 (800+1), …
  `&vTab[6]` vaut 806 (800+6), donc :

- `&vTab[i] == vTab + i`
  (arithmétique des pointeurs)

- **`vTab[i]`** `== *&vTab[i] == *(vTab+i)`

- Juste du « sucre syntaxique »…

4.1 Équivalence pointeur ←→ tableau

- `char vTab[] = "Bonjour";`
- `&vTab[i] == vTab + i`
  (arithmétique des pointeurs)
- **`&vTab[0]`** `== vTab + 0 ==` **`vTab`**
- `char * p = &vTab[0];` ou plus simplement :
  `char * p = vTab;`

```
printf( "%s", p );
printf( "%s", vTab );
```

4.1 Arithmétique des pointeurs

- `double vTab[] = {1.1,…,6.6};`
  `&vTab[0] == vTab` : 400
  `&vTab[1] == vTab+1` : 401 ?

- non !
  `&vTab[1]` : 408,
  `&vTab[2]` : 416 etc.

- Donc `double[] ≠ int[]`
- Donc `double* ≠ int*`

QUIZZ

- `double vTab[] = {1.1,…,6.6};`
  `double* vP;`

- `vP = ?;`

A **vTab**	**D** **autre**
B **&vTab**	**C** ***vTab**

QUIZZ

- `double vTab[] = {1.1,…,6.6};`
  `double* vP = vTab;`

- `1) vP[2] = 7.7;`
  `2) *vTab = 8.8;`

A **seulement 1**	**D** **aucune**
B **seulement 2**	**C** **1 et 2**

4.2 Différences pointeur ←→ tableau

- `char* v;`
  réserve 4 octets dans lesquels on pourra stocker n'importe quelle adresse de `char` mais on ne peut stocker AUCUN caractère !
- `char v[10];`
  réserve 14 octets dont 4 contenant l'adresse des 10 octets dans lesquels on pourra stocker 10 caractères
- `char* v;`
  déclare un pointeur variable `v=&tab[2];`
- `char v[10];`
  déclare un pointeur constant **`v= /!\`**

QUIZZ

- ```
  int p1[] = {1,2,3};
  int * p2 = p1 + 2;
  ```

- ```
  1) p1 = p2;
  2) p2 = p1;
  ```

A **seulement 1**	**D** **aucune**
B **seulement 2**	**C** **1 et 2**

5. Les pointeurs de pointeurs

Dans le `main` :

```
char vMot[] = "Bonjour";
char * vPtrChar = vMot;
dernierChar( &vPtrChar );
printf( "%c\n", *vPtrChar );

void dernierChar( char* * pPS )
{
   *pPS = *pPS + strlen( *pPS ) - 1;
}
```

6. Un type pointeur

Dans le `main` :

```
typedef char * PtrChar;
char vMot[] = "Bonjour";
PtrChar vPtrChar = vMot;
dernierChar( &vPtrChar );
printf( "%c\n", *vPtrChar );
void dernierChar( PtrChar * pPS )
{
   *pPS = *pPS + strlen( *pPS ) - 1;
}
```