Liste d'exercices 8 : les pointeurs (1 séance)

Notions à considérer pour pouvoir faire les exercices

Accès directs en mémoire versus accès indirects en mémoire :

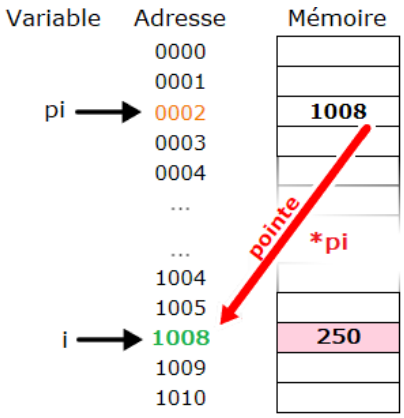
Le pointeur n'est rien de plus qu'un intermédiaire pour atteindre une variable dans laquelle est stockée une donnée.

Exemple :

```
int i, *pi = &i;
```

Langage C	*Assembleur*	*Langage C*	*Assembleur*
`i = 250;`	`mov i, 250`	`*pi = 250;`	`mov ebx, pi` `mov DWORD PTR [ebx], 250`

Dans ce diagramme, l'adresse en mémoire de la variable i est 1008 (choix arbitraire). Cette adresse a été copiée dans le pointeur pi lors de sa déclaration. Ensuite, avec ***pi = 250**, on fait **pointer pi vers la variable i** dans laquelle est alors copiée la valeur 250.



Dès qu'un accès indirect en mémoire a été réalisé avec un pointeur pour obtenir la valeur se trouvant dans la variable vers laquelle il pointe, les règles à appliquer ensuite concernant les autres traitements (priorité des opérateurs, fonctionnement de chaque opérateur, castings implicites, etc.) restent inchangées par rapport à ce qui a été vu jusqu'à présent dans ce cours.

Exemple:

```
char   b, *pb = &b;
short  s, *ps = &s;
int    i, *pi = &i;
float  f, *pf = &f;
double d, *pd = &d;
```

Langage C	*Assembleur*	*Langage C*	*Assembleur*

`b = b * 2;`	`movsx eax, b`
	`imul  eax, 2`
	`mov   b, al`
		`*pb = *pb * 2;`	`mov   ebx, pb`
			`movsx eax, BYTE PTR [ebx]`
			`imul  eax, 2`
			`mov   BYTE PTR [ebx], al`

`s = s * 2;`	`movsx eax, s`
	`imul  eax, 2`
	`mov   s, ax`
		`*ps = *ps * 2;`	`mov   ebx, ps`
			`movsx eax, WORD PTR [ebx]`
			`imul  eax, 2`
			`mov   WORD PTR [ebx], ax`

`i = i * 2;`	`mov   eax, i`
	`imul  eax, 2`
	`mov   i, eax`
		`*pi = *pi * 2;`	`mov   ebx, pi`
			`mov   eax, DWORD PTR [ebx]`
			`imul  eax, 2`
			`mov   DWORD PTR [ebx], eax`

`f = f * 2;`	`movss    xmm0, f`
	`mov      eax, 2`
	`cvtsi2ss xmm1, eax`
	`mulss    xmm0, xmm1`
	`movss    f, xmm0`
		`*pf = *pf * 2;`	`mov      ebx, pf`
			`movss    xmm0, DWORD PTR [ebx]`
			`mov      eax, 2`
			`cvtsi2ss xmm1, eax`
			`mulss    xmm0, xmm1`
			`movss    DWORD PTR [ebx], xmm0`

`d = d * 2;`	`movsd    xmm0, d`
	`mov      eax, 2`
	`cvtsi2sd xmm1, eax`
	`mulsd    xmm0, xmm1`
	`movsd    d, xmm0`
		`*pd = *pd * 2;`	`mov      ebx, pd`
			`movsd    xmm0, QWORD PTR [ebx]`
			`mov      eax, 2`
			`cvtsi2sd xmm1, eax`
			`mulsd    xmm0, xmm1`
			`movsd    QWORD PTR [ebx], xmm0`

`d = (double)i / 3;`	`cvtsi2sd xmm0, i`
	`mov      eax, 3`
	`cvtsi2sd xmm1, eax`
	`divsd    xmm0, xmm1`
	`movsd    d, xmm0`
		`d = (double)*pi / 3;`	`mov      ebx, pi`
			`cvtsi2sd xmm0, DWORD PTR [EBX]`
			`mov      eax, 3`
			`cvtsi2sd xmm1, eax`
			`divsd    xmm0, xmm1`
			`movsd    d, xmm0`

`i = (int)d / 3;`	`cvttsd2si eax, d`
	`cdq`
	`mov       ebx, 3`
	`idiv      ebx`
	`mov       i, eax`
		`i = (int)*pd / 3;`	`mov       ebx, pd`
			`cvttsd2si eax, QWORD PTR [ebx]`
			`cdq`
			`mov       ebx, 3`
			`idiv      ebx`
			`mov       i, eax`

`i = (float)d / 3;`	`cvtsd2ss  xmm0, d`
	`mov       eax, 3`
	`cvtsi2ss  xmm1, eax`
	`divss     xmm0, xmm1`
	`cvttss2si eax, xmm0`
	`mov       i, eax`
		`i = (float)*pd / 3;`	`mov       ebx, pd`
			`cvtsd2ss  xmm0, QWORD PTR [ebx]`
			`mov       eax, 3`
			`cvtsi2ss  xmm1, eax`
			`divss     xmm0, xmm1`
			`cvttss2si eax, xmm0`
			`mov       i, eax`

`d = i / 2.0;`	`cvtsi2sd  xmm0, i`
	`divsd     xmm0, d2`
	`movsd     d, xmm0`
		`d = *pi / 2.0;`	`mov       ebx, pi`
			`cvtsi2sd  xmm0, DWORD PTR [EBX]`
			`divsd     xmm0, d2`
			`movsd     d, xmm0`

Exercices

Écrivez la séquence d'instructions en assembleur qui correspond à chaque instruction d'affectation en langage C suivante :

```
char   b = -20, *pb = &b;
int    i = 1317, *pi = &i;
float  f = 2452.578, *pf = &f;
double d = 7983.14, *pd = &d;
```

- `*pd = b + *pf * 82.195;` `// d = 201569.63891723630`
- `f = *pi * (float)*pd;` `// f = 10513796.0`
- `i = -((float)*pi) * (2 + f - 5.2 * *pb);` `// i = -3369647`
- `f = (int)*pd / (*pb - 36);` `// f = -142.0`
- `*pi = (float)*pd * 2 + ++(*pf) - 5 * b;` `// i = 18519; f = 2453.57788`
- `*pd = ((int)*pd / 100) + (--(*pf) / 20);` `// d = 201.57888793945313; f = 2451.57788`
- `d = -((double)b + (int)*pd * i) / --(*pb);` `// d = 500647.14285714284; b = -21`
- `f = (int)(-97 - --(*pf) - - --(*pb) * d + 46.67);` `// f = -170147.0; b = -21`
- `d = (int)(*pd)++ * 3 + (double)-i / *pf;` `// d = 23949.463013994264`
- `*pi = (int)(*pf + 0.5 * i) | 0xff + *pb;` `// i = 3311`
- `*pf = (*pi * 0xff & 0xfff0) / (*pd - 4278.45);` `// f = 2.19829464`

Rappel : les valeurs numériques dans les expressions qui ont une partie décimale, par exemple la valeur 5.2, sont du type double, tandis que les valeurs numériques qui n'ont aucune partie décimale, par exemple la valeur 3, sont du type int.