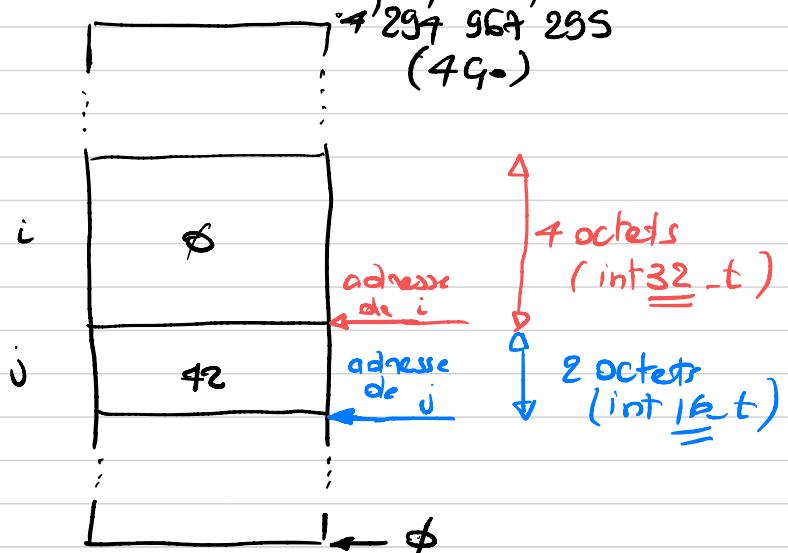


Info2 - Pointeurs et Allocation mémoire

29.04.2021

Labo 04. → date du rendu : Dimanche 21 Avril 23^h

```
int32_t i = 0;
int16_t j = 42;
```



Pour connaître l'adresse d'une variable (non tableau) : & var
 Pour afficher (printf) une adresse, le format : %p

TD 20210329

- 1) créez i et j tel qu'indiqué ci-dessus
- 2) affichez les adresses de i et de j
- 3) exécutez plusieurs fois ./app et observez le résultat.

15^h

```
int main(int argc, const char *argv[])
{
    int32_t i = 0;
    int16_t j = 42;

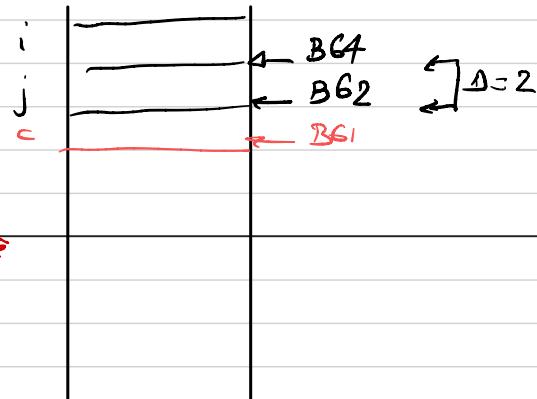
    printf("adress of i: %p\n", &i);
    printf("adress of j: %p\n", &j);
    return 0;
}
```

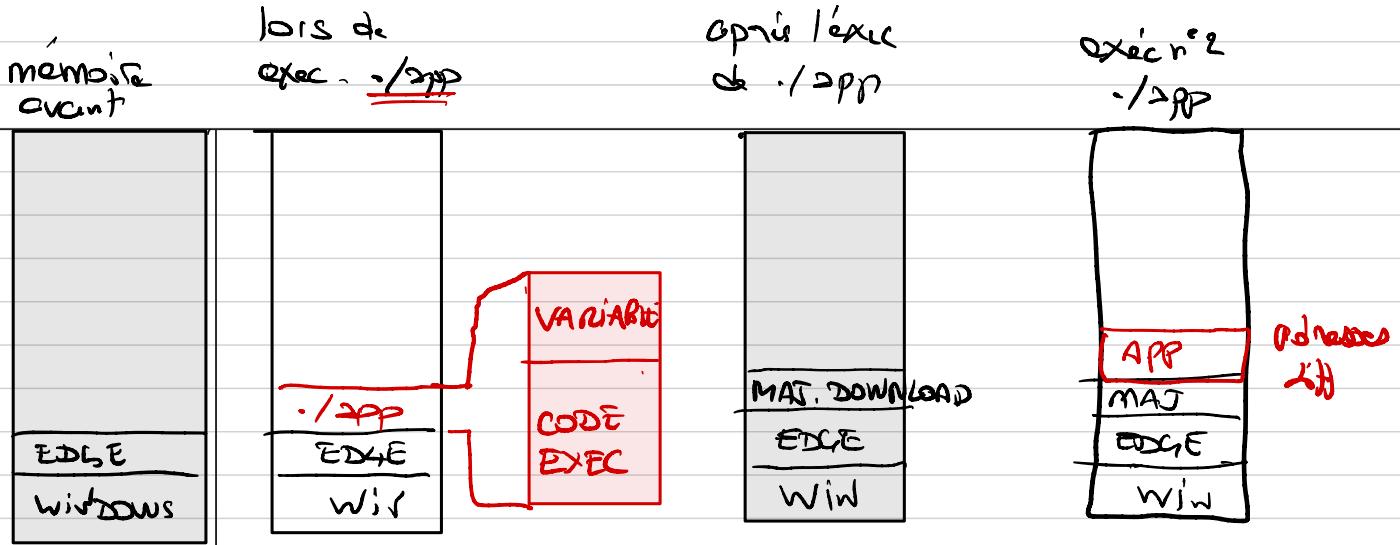
```
→ TD20210329 git:(main) ✘ ./app
adress of i: 0x7ffc1c78db64
adress of j: 0x7ffc1c78db62
→ TD20210329 git:(main) ✘ ./app
adress of i: 0x7ffc94b1dc94
adress of j: 0x7ffc94b1dc92
→ TD20210329 git:(main) ✘ ./app
adress of i: 0x7ffc4fb8dbf4
adress of j: 0x7ffc4fb8dbf2
```

On ne peut pas (jamais) utiliser cba <= 64 & en dur (de manière numérique)
 l'adresse d'un variable. On doit passer par &

~~scanf ("%hd", &i);~~

scanf ("%hd", &j);





Création d'une variable : type identificateur = valeur initiale ;
(simple) uint32_t i = \$;

Création d'une variable qui contient une adresse.

exemple d'une variable qui contient l'adresse d'une variable du type uint32_t :

uint32_t * pi = &i ;
 typ. "pointeur sur un uint32_t"

pi : variable
 elle contient une adresse (*)
 cette adresse est celle d'une variable de type uint32_t

elle contient l'adresse de la variable i

! TRES important : de part et d'autre du symbole égal on doit avoir le même type de donnée

uint32_t * pi = &i ;
 typ. pointeur (donne une adresse) sur un type uint32_t

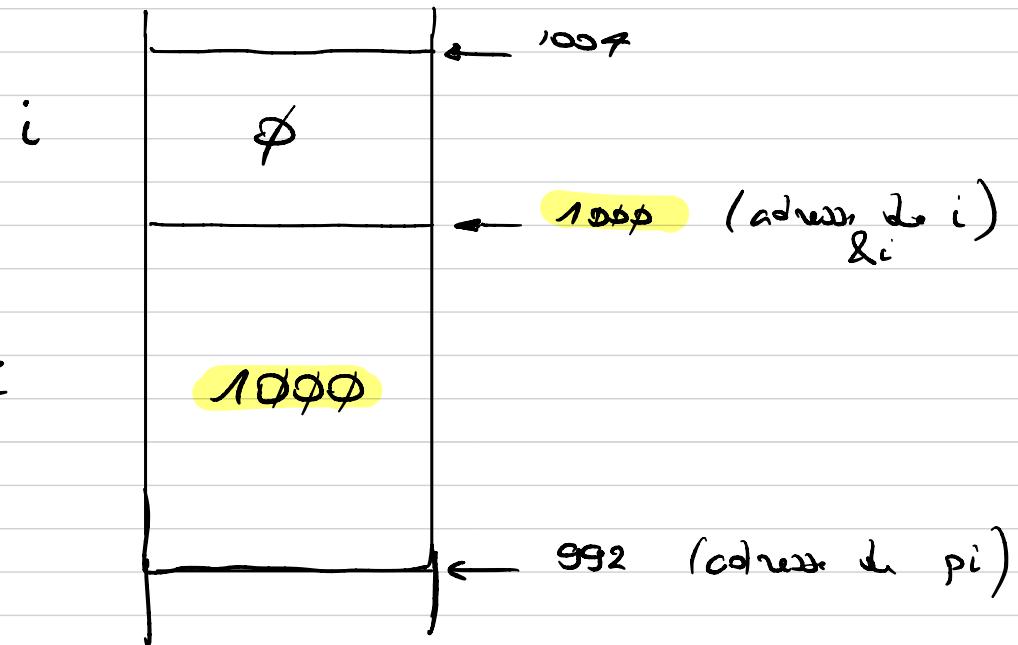
l'adresse de i
 l'adresse d'un type uint32_t

```
int main()
{
```

adresse : 1000 octets

uint32_t c = 0;

uint32_t * pi = &i;

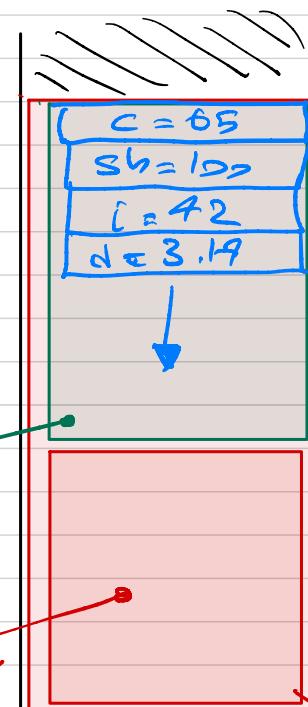


```
int main()
```

```
char c = 65;
short sh = 100;
int i = 42;
double d = 3.14
```

ZONE DES DONNÉES

CODE EXECUTABLE



"haut" de la zone des données (pile) stack
Taille Max pour var !

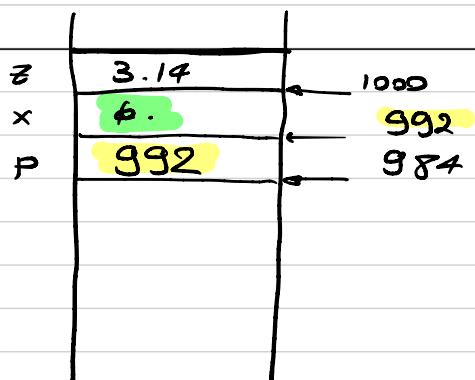
ZONE MÉMOIRE DÉDIÉ À ./app

```
double z = 3.14;  
double x = 0.;
```

```
double *p;
```

~~P=x;~~

$\text{(!) 2 types } \neq \text{ mêmes}$
 $\text{du signe } =$



p: type double
x: double



$P = \&x;$

\downarrow \downarrow
adresse de x
adresse d'une var. de
type double.

double *
= adresse d'un
double.

scanf ("%pf", &z);

\downarrow l'adresse de la variable z
(1000 dans notre exemple)

fonction scanf: 1 chaîne de caractères: "%pf"
l'adresse d'une variable à mettre à jour

Pour accéder au contenu de l'adresse on utilise
le symbol. *

lire le contenu: printf ("%pf", *p);

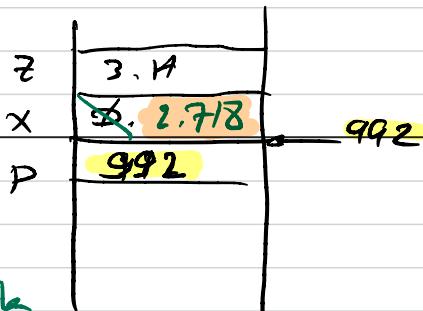
\hookrightarrow affiche le contenu de l'adresse stockée dans p.

p vaut 992
*p vaut 0.

à l'adresse 992 il y a
une valeur de type double
qui vaut 0.



$*p = 2.718;$
type double double



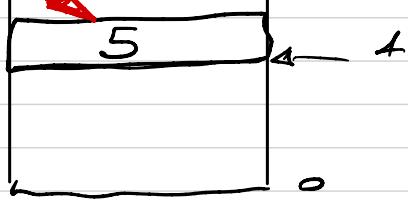
Si on connaît l'adresse d'une variable
on peut la modifier grâce à un pointeur.

	<i>i</i>	p^i	
<code>int i;</code>			
<code>int * pi;</code>			
<code>pi = &i;</code>	?	<u>1000</u>	
<code>i = 1;</code>	1	1000	
<code>*pi = 2;</code>	2	1000	
<code>*&i = 3;</code>	3	1000	
<code>pi = 4;</code>	3	4	
<code>*pi = 5;</code>	3	4	

Remplissez le tableau au fur et à mesure de l'exécution des quelques lignes

Ecriture dans une zone "douceur"

"Segmentation Fault"



16 / 13

$\ast \underline{\& i} = 3;$

\ast adresse d'un int } ne s'emploie pas...

contenu de l'adresse d'un int = int

$\underline{pi} = 4$

adresse d'un int } int

2 types \neq contenu du même =

⚠️ CA VA GÉNÉRER UN WARNING.

WARNING : TYPES \neq MAIS UNE ADRESSE EST UN ENTIER \Rightarrow CA COMPILE