

Programme:

lire un fichier WAV : durée du son ?

inconnue lors de la compilation
 \Rightarrow créer un tableau avec **malloc** pour accueillir les données

ex: 10' max avec $F_s = 48000$ Hz et stéréo 16b.

taille en octets: $10 \times 60 \times 48000 \times 2 \times 2 = 115200000$!
 Chacun $t[115200000]$;

||| Un tableau à taille fixe n'est pas compatible avec le traitement des données dont la taille n'est pas connue à priori;

Solution

1) Estimer la quantité mémoire nécessaire
 (lecture d. l'entête du fichier wav)

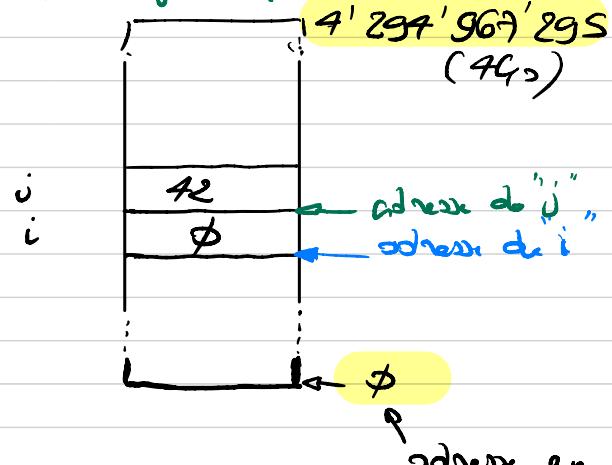
2) Demander au système de nous réservé un emplacement mémoire correspondant

Pointeurs

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

// adresse de i & i
// c'est un entier

// affichage: %p



On ne peut pas fixer en dur des valeurs
 d'adresses des variables dans des pointeurs.

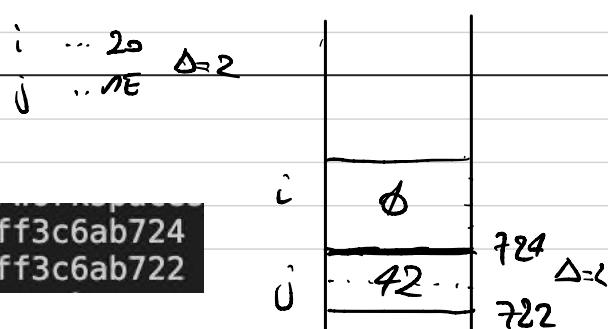
TD2021/0325

- 1) créer i et j telle qu'il adapte
- 2) d'afficher à l'écran les adresses de ces variables

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

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

```
adress of i : 0x7fff3c6ab724
adress of j : 0x7fff3c6ab722
```



Variable $\text{int32_t } i = \phi;$

Pour manipuler l'adresse de i dans une variable il faut définir une variable dont le contenu est l'adresse de i

$\underbrace{\text{int32_t}}_{\text{type}} \underbrace{i}_{\substack{\text{identificateur} \\ | \\ \text{valeur de } i}} = \phi;$

$\underbrace{\text{int32_t } *}_{\substack{\text{type "pointeur} \\ \text{sur un int32_t}}} \underbrace{pi}_{\substack{\text{identificateur} \\ | \\ \text{valeur de pi}}} = \underbrace{\&i}_{\text{valeur de pi}}$

* : pi contient l'adresse d'un int32_t

& : l'adresse de la variable " i " qui est du type int32_t

$\text{int32_t } * pi = \underbrace{\&i}_{\substack{\text{type adresse} \\ \text{d'un int32_t}}} ;$

Même type de donnée de part et d'autre du signe \Rightarrow OK

$\text{double } z = 3.14;$
 $\text{double } x = \phi;$
 $\text{double } *p;$

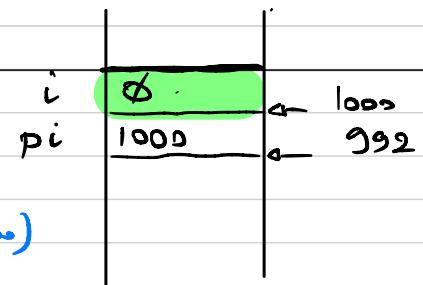
z	3.14	1000
x	$\phi.$	998
p	1000	984

~~$p = x;$~~ $p: \text{double } *$

$x: \text{double}$

✓ $p = \&z;$ $p: \text{double } *$ (adresse d'un double)
 $\&z:$ adresse de z ("")

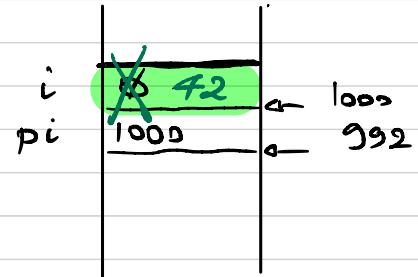
`int32_t i = 0;`
`int32_t *pi = &i;`



// pi contient l'adresse de i (1000)

`*pi = 42;`

// *pi : c'est le contenu de la case 1000



Fonction

Passage par valeur

`main() {`

`i = 42;`

`f(i); // f(42),`

`void f(int j) { // j ← 42
j++; j ← 43`

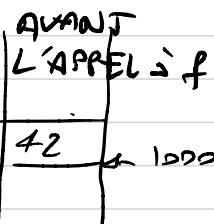
? p

Passage par adresse

`main() {`

`i = 42;`

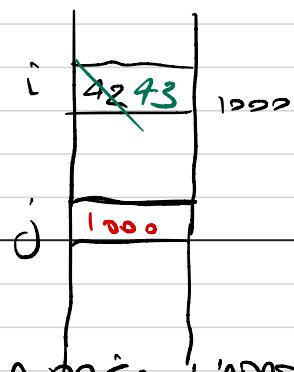
`f(&i);`



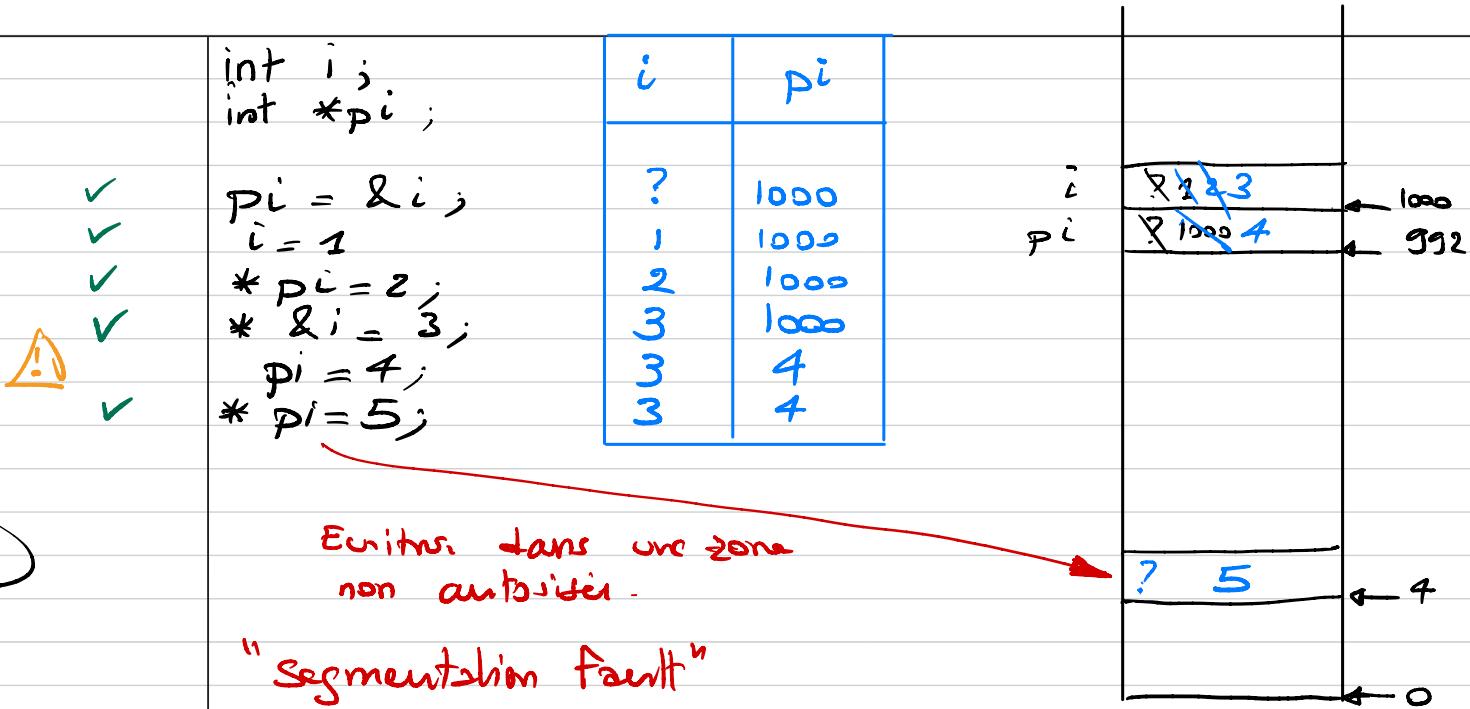
`void f(int *j) {`

`*j = *j + 1;`

}



`scanf ("%d", &i);`



$*\pi^i = 2$
int int

$*\&i = 3$ ← ne s'applique pas =)

adresse
d'un int

adresse d'un int
contenu de l'int
contenu de i

$\pi^i = 4$ (!) types \neq
int* int

une adresse est un
entier.

→ générer un warning Mais le compil

~~* pi=5~~
int int

- Suite du TD:
- ajouter les lignes du code ci-dessus
 - observer le warning lors de la compilation
 - que se passe-t-il lors de l'exécution ?

Résumé

```
int main(int argc, const char *argv[])
{
    int i;
    int *pi = &i;
    printf("i : %6d pi : %p\n", i, pi);
    i = 1;
    printf("i = 1; // i : %6d pi : %p\n", i, pi);
    pi = &i;
    printf("pi = &i; // i : %6d pi : %p\n", i, pi);
    *pi = 2;
    printf("*pi = 2; // i : %6d pi : %p\n", i, pi);
    *&i = 3;
    printf("*&i = 3; // i : %6d pi : %p\n", i, pi);
    pi = 4;
    printf("pi = 4; // i : %6d pi : %p\n", i, pi);
    *pi = 5;
    printf("*pi = 5; // i : %6d pi : %p\n", i, pi);

    return 0;
}
```

Segmentation fault

$\&$ variable : adresse de la variable

type * variable : variable est un pointeur sur le type

variable contient l'adresse d'un information à type type

* variable : contenu de l'adresse binarie à variable

