Avertissement

Ce document accompagne le TP associé. Il ne fera l'objet d'aucune évaluation.

Crédits

- Un zeste de savoir.
- ≪ Langage C ≫ (Claude Delannoy) Eyrolles.

Déclaration

- Un *pointeur de fonction* se définit à l'aide d'une syntaxe mélangeant celle des pointeurs sur tableau et celles des prototypes de fonction. Voici la définition d'un pointeur sur une fonction retournant un int et attendant un int comme argument.

```
1       int (*pf)(int);
2
```

Déclaration

- Un *pointeur de fonction* se définit à l'aide d'une syntaxe mélangeant celle des pointeurs sur tableau et celles des prototypes de fonction. Voici la définition d'un pointeur sur une fonction retournant un int et attendant un int comme argument.

```
1       int (*pf)(int);
2
```

- Comme les pointeurs sur tableau, il faut d'entourer le symbole `*` et l'identificateur de parenthèses afin d'éviter que cette déclaration ne soit vue comme un prototype et non comme un pointeur de fonction.

Déclaration

- Un *pointeur de fonction* se définit à l'aide d'une syntaxe mélangeant celle des pointeurs sur tableau et celles des prototypes de fonction. Voici la définition d'un pointeur sur une fonction retournant un int et attendant un int comme argument.

```
1       int (*pf)(int);
2
```

- Comme les pointeurs sur tableau, il faut d'entourer le symbole `*` et l'identificateur de parenthèses afin d'éviter que cette déclaration ne soit vue comme un prototype et non comme un pointeur de fonction.
- Le type de retour, le nombre d'arguments et leur type doivent également être spécifiés.

Ecritures équivalentes

- Observons :

```
1  int (*pf)(int);
2  pf = &f;
3  pf = f;// transformé en &f
```

Ecritures équivalentes

- Observons :

```
1  int (*pf)(int);
2  pf = &f;
3  pf = f;// transformé en &f
```

- Les deux écritures lignes 2 et 3 sont identiques :

Ecritures équivalentes

- Observons :

```
1  int (*pf)(int);
2  pf = &f;
3  pf = f;// transformé en &f
```

- Les deux écritures lignes 2 et 3 sont identiques :
  - un identificateur de fonction isolé comme en ligne 3 est converti par le compilateur en un pointeur sur cette fonction.

Ecritures équivalentes

- Observons :

```
1  int (*pf)(int);
2  pf = &f;
3  pf = f;// transformé en &f
```

- Les deux écritures lignes 2 et 3 sont identiques :
  - un identificateur de fonction isolé comme en ligne 3 est converti par le compilateur en un pointeur sur cette fonction.
  - en ligne 2, l'identificateur de fonction est l'opérande de &. Il est laissé tel quel.

Ecritures équivalentes

- Observons :

```
1  int (*pf)(int);
2  pf = &f;
3  pf = f;// transformé en &f
```

- Les deux écritures lignes 2 et 3 sont identiques :
  - un identificateur de fonction isolé comme en ligne 3 est converti par le compilateur en un pointeur sur cette fonction.
  - en ligne 2, l'identificateur de fonction est l'opérande de `&`. Il est laissé tel quel.
- Au final les deux écritures correspondent bien à un pointeur sur `f`.

Déréférencement

L'opérateur `*` et l'identificateur de fonction doivent être entre parenthèses. Les arguments à passer au pointeur de fonction déférérencé suivent la syntaxe habituelle

```c
#include <stdio.h>
// d'après zeste de savoir
int triple(int a)
{
    return a * 3;
}

int main(void)
{
    int (*pt)(int) = &triple;

    printf("%d.\n", (*pt)(3));// affiche 9
    return 0;
}
```

Déréférencement (suite)

- Le déréférencement avec `*` n'est pas nécessaire. En effet, un identificateur de fonction non opérande de `&` est converti en un pointeur de fonction.
- Sous les conventions du code précédent, l'expression `triple(3)` est converti en un pointeur de fonction auquel on passe l'agument 3.
- L'expression `(*pt)(3)` est équivalent à un identificateur de fonction appliquée à 3. cet identificateur est ensuite reconverti en un pointeur de fonction auquel on passe l'argument 3 ! !

Passagee en argument

Les fonctions sont passées en argument par le biais d'un pointeur de fonction

```c
#include <stdio.h>
// zeste de savoir
 int triple(int a)
{
    return a * 3;
}

 int quadruple(int a)
{
    return a * 4;
}

 void affiche(int a, int (*pf)(int))
{
    printf("%d.\n", (*pf)(a));
}

```