進階資料型態定義

丁培毅

以 *typedef* 重新定義型態名稱

- Simple rule: *typedef* original_type_name new_type_name;

```
typedef unsigned long ulong;
ulong x; // equivalent to unsigned long x;
```

- More general rule: *typedef* type definition of new_type_name;

```
typedef int IntAry[20];
IntAry y[30]; // equivalent to int y[30][20];
```

```
typedef double (*(*FP)( ))[10];
FP fp; // equivalent to double (*(*fp)())[10];
meaning: “a function pointer fp to a function that takes no argument and
         returns a pointer to a 10-element array of doubles”
Equivalent and more self-explaining definitions:
    typedef double DoubleArray[10];
    typedef DoubleArray *PtrDoubleArray;
    typedef PtrDoubleArray (*FP)();
```

以 *typedef* 重新定義型態名稱

- You can also define multiple new type names in one *typedef* statement, just like you define several variables in one definition.

```
typedef struct
{
   int x;
   int y;
} Point, *PtrPoint;
Point point;         // equivalent to struct { int x; int y; } point;
PtrPoint ptrPoint; // equivalent to struct { int x; int y; } *ptrPoint;
```

複雜的資料型態範例

- Examples:

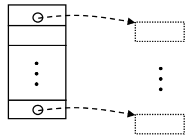
```
int *x;
int *x[10];
int (*x)[10];
int (**x)[10];
int *(**x)[10];
int x[10][20][30];
void (*funcPtr)();
void *func();  // definition of a function
void (*signal(int, void(*)(int)))(int);  // definition of a function
void *(*(*fp1)(int))[10];
float (*(*fp2)(int, int, float))(int);
double (*(*(*fp3)())[10])();
int (*(*f4())[10])();
```

Using *typedef* can simplify these definitions

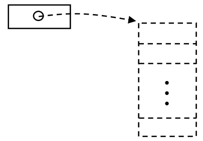
陣列與指標型態

✧ Two simplest examples first:

int *x[10]; // 10-element ARRAY of (PTR to integer)



int (*x)[10]; // PTR to (10-element ARRAY of integers)

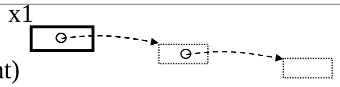


TYPE [n]	means "n-element ARRAY of TYPE"
TYPE *	means "PTR to TYPE"

[ ] has higher precedence than *, ( ) can change the priority

陣列與指標型態 (cont'd)

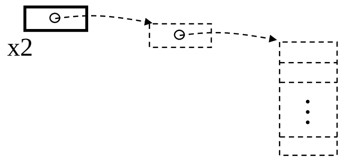
✧ Some more

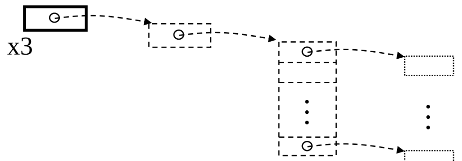


int **x1; // PTR to (PTR to int)

int (x2)[10];** // PTR to (PTR to (10-element ARRAY of int))

int *(x3)[10];** // PTR to (PTR to (10-elem ARY of (PTR to int)))





✧ Use typedef to simplify the definition

```
typedef int *IPTR;
IPTR *x1;
typedef int IARY[10];
typedef IARY *PTRIARY;
PTRIARY *x2;
```

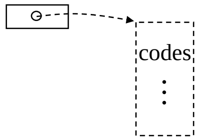
```
typedef IPTR IPTRARY[10];
typedef IPTRARY *PTR_IPTRARY;
PTR_IPTRARY *x3;
```

Note: sizeof(x1)=4, sizeof(x2)=4,
sizeof(x3)=4

函式指標型態

✧ Function pointers

void (*funcPtr)(); // PTR to a function that takes no argument and returns void



✧ Real example:

void (*(*fp)(int, void(*)(int)))(int);

// fp is a PTR to a function that takes two arguments, an int, (a function pointer that takes one int argument and returns void), and returns (a function pointer that takes one int argument and returns void)

Equivalently,

typedef void (*sig_t)(int);
typedef sig_t (*FP)(int, sig_t);
FP fp;

陣列與函式指標型態

✧ Ex: PTR to a function that takes an int argument and returns a PTR to (10-element ARRAY of (PTR to void))

void *(*(*fp1)(int))[10];

✧ Ex: PTR to a function that takes three arguments: int, int, float and returns (PTR to a function that takes an int argument and returns float)

float (*(*fp2)(int, int, float))(int);

✧ Ex: PTR to a function that takes no argument, returns (PTR to (10-element ARRAY of (PTR to a function that takes no argument and returns double)))

double (*(*(*fp3)())[10])();

✧ Ex: function that takes no argument, returns (PTR to (10-element ARRAY of (PTR to function that takes no argument and returns int)))

int (*(*f4())[10])();