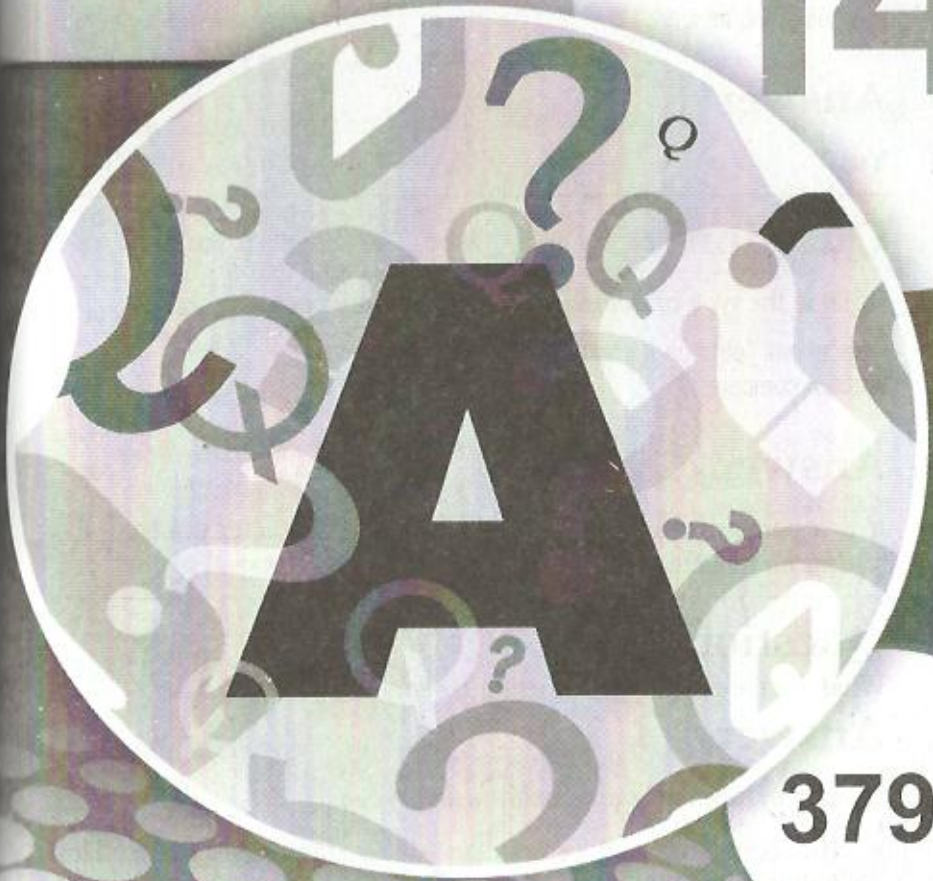
# Subtleties of typedef

# 14



## Question 14.1

Are the properties of *i*, *j* and *x*, *y* in the following program same? [Yes/No]

```
typedef unsigned long int uli ;
uli i, j ;
unsigned long int x, y ;
```

## Answer

Yes

---

## Question 14.2

What is the type of *compare* in the following code segment?

```
typedef int ( *ptrtofun )( char *, char * ) ;
ptrtofun compare ;
```

## Answer

It is a pointer to function that receives two character pointers and returns an integer.

---

## Question 14.3

What are the advantages of using *typedef* in a program?

## Answer

There are three main reasons for using *typedef*s:

- It makes writing of complicated declarations a lot easier. This helps in eliminating a lot of clutter in the program.

- It helps in achieving portability in programs. That is, if we use *typedef*s for data types that are machine-dependent, only the *typedef*s need change when the program is moved to a new machine platform.

- It helps in providing a better documentation for a program. For example, a node of a doubly linked list is better understood as *ptrtolist* rather than just a pointer to a complicated structure.



---

## Question 14.4

Is there any difference in the *#define* and the *typedef* in the following code? If yes, what?

```
typedef char * string_t ;
#define string_d char *
string_t s1, s2 ;
string_d s3, s4 ;
```

## Answer

In these declarations, *s1*, *s2*, and *s3* are all treated as *char* *, but *s4* is treated as a *char*, which is probably not the intention.

---

## Question 14.5

Which of the following will be the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    typedef int LONG ;
    LONG a = 4 ;
    LONG b = 68 ;
    float c = 0 ;
    c = b ;
    b += a ;
```

```
    printf ( "%d ", b ) ;
    printf ( "%f\n", c ) ;
    return 0 ;
}
```

A. 72 68.000000
B. 72.00000000 68
C. 68.000000 72.000000
D. 68 72.000000

## Answer

A

---

## Question *14.6*

Which of the following will be the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    enum color { red, green, blue } ;
    typedef enum color mycolor ;
    mycolor m = red ;
    printf ( "%d\n", m ) ;
    return 0 ;
}
```

A. 1
B. 0
C. 2
D. red



## Answer

B

---

## Question *14.7*

Which of the following will be the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    typedef int arr[ 5 ] ;
    arr iarr = { 1, 2, 3, 4, 5 } ;
    int i ;
    for ( i = 0 ; i < 4 ; i++ )
        printf ( "%d ", iarr[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

A. 1 2 3 4
B. 1 2 3 4 5
C. No output
D. Error: Cannot use typedef with an array

## Answer

A

---

## Question *14.8*

Which of the following will be the correct output for the program given below?

```
#include <stdio.h>
```

```
int main( )
{
    typedef float f ;
    static f *fptr ;
    float fval = 90 ;
    fptr = &fval ;
    printf ( "%f\n", *fptr ) ;
    return 0 ;
}
```

A. 9
B. 0
C. 90.000000
D. 90

## Answer

C

---

## Question *14.9*

*typedefs* have the advantage that they obey scope rules, that is, they can be declared local to a function or a block whereas *#defines* always have a global effect. [True/False]

## Answer

True

---

## Question *14.10*

Point out the error in the following declaration and suggest at least three solutions for it.

```
typedef struct
{
    int data ;
    NODEPTR link ;
} *NODEPTR ;
```



## Answer

A *typedef* declaration cannot be used until it is defined, and in our example it is not yet defined at the point where the *link* field is declared.

We can fix this problem in three ways:

(1) Give the structure a name, say *node* and then declare the *link* field as a simple *struct node* * as shown below:

```
typedef struct node
{
    int data ;
    struct node *link ;
} *NODEPTR ;
```

(2) Keep the *typedef* declaration separate from the structure definition:

```
struct node
{
    int data ;
    struct node *link ;
} ;
typedef struct node * NODEPTR ;
```

(3) Precede the structure declaration with *typedef*, so that you can use the NODEPTR *typedef* when declaring the *link* field:

```
typedef struct node *NODEPTR ;
struct node
{
    int data ;
    NODEPTR link ;
```

```
};
```

## Question 14.11

In the following code snippet can we declare a new *typedef* named *ptr* even though *struct employee* has not been completely declared while using the *typedef*? [Yes/No]

```
typedef struct employee *ptr ;
struct employee
{
    char name[20] ;
    int age ;
    ptr next ;
} ;
```

## Answer

Yes

## Question 14.12

There is an error in the following declarations. Can you rectify it?

```
typedef struct
{
    int data1 ;
    BPTR link1 ;
} *APTR ;

typedef struct
{
    int data2 ;
    APTR link2 ;
} *BPTR ;
```



## Answer

The problem with the code is the compiler doesn't know about BPTR when it is used in the first structure declaration. We are violating the rule that a *typedef* declaration cannot be used until it is defined, and in our example it is not yet defined at the point where the *link1* field is declared.

To avoid this problem we can define the structures as shown below:

```
struct a
{
    int data1 ;
    struct b *link1 ;
} ;
struct b
{
    int data2 ;
    struct a *link2 ;
} ;

typedef struct a *APTR ;
typedef struct b *BPTR ;
```

The compiler can accept the field declaration *struct b *ptr1* within *struct a*, even though it has not yet heard of *struct b* (which is "incomplete" at that point). Occasionally, it is necessary to precede this couplet with the line

```
struct b ;
```

This empty declaration masks the pair of structure declarations (if in an inner scope) from a different *struct b* in an outer scope. After declaring the two structures we can then declare the *typedefs* separately as shown above.

Alternatively, we can also define the *typedef*s before the structure declaration, in which case you can use them when declaring the *link* pointer fields as shown below:

```
typedef struct a *APTR ;
typedef struct b *BPTR ;
struct a
{
    int data1 ;
    BPTR link1 ;
} ;
struct b
{
    int data2 ;
    APTR link2 ;
} ;
```

## Question *14.13*

What is *x* in the following program?

```
#include <stdio.h>
int main( )
{
    typedef char ( * ( * arrfptr[3] ) ( ) ) [10] ;
    arrfptr x ;
    return 0 ;
}
```

## Answer

Here *x* is an array of three function pointers. Each function pointer points to a function that returns a pointer to an array of 10 chars.



## Question *14.14*

What will be the output of the following program?

```
typedef struct data
{
    int x ;
    sdata *b ;
} sdata ;
```

## Answer

Error: 'Declaration missing ;'. Since the type name **sdata** is not known at the point of declaring the structure.

## Question *14.15*

Which of the following will be the correct output for the program given below?

```
#include <stdio.h>
typedef struct error { int warning, err, exception ; } ERROR ;
int main( )
{
    ERROR e ;
    e.err = 1;
    printf ( "%d\n",e.err ) ;
    return 0 ;
}
```

A. 0
B. 1
C. 2
D. Error

## Answer

B

## Question *14.16*

What do the following declarations mean?

```
typedef char *pc ;
typedef pc fpc( ) ;
typedef fpc *pfpc ;
typedef pfpc fpfpc( ) ;
typedef fpfpc *pfpfpc ;
pfpfpc a[N] ;
```

## Answer

*pc* is a pointer to *char*.

*fpc* is function returning pointer to *char*.

*pfpc* is pointer to a function returning pointer to *char*.

*fpfpc* is a function returning pointer to a function returning pointer to *char*.

*pfpfpc* is a pointer to function returning pointer to a function returning pointer to *char*.

*pfpfpc a[N]* is an array of *N* pointers to functions returning pointers to functions returning pointers to characters.

## Question *14.17*

How will you define *a[N]* in 14.16 above without using *typedef*?



## Answer

```
char *(*(*a[N])( ))( ) ;
```

## Question *14.18*

Improve the following code using *typedef*.

```
struct node
{
    int data1 ; float data2 ;
    struct node *left ;
    struct node *right ;
} ;
struct node *ptr ;
ptr = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
```

## Answer

```
typedef struct node * treeptr ;
typedef struct node
{
    int data1 ;
    float data2 ;
    treeptr *left ;
    treeptr *right ;
} treenode ;
treeptr ptr ;
ptr = ( treeptr ) malloc ( sizeof ( treenode ) ) ;
```

## Question *14.19*

Is the following declaration acceptable? [Yes/No]

```
typedef long no, *ptrtono ;
no n ;
ptrtono p ;
```

## Answer

Yes

---

## Question *14.20*

In the following code what is constant, p or the character it is pointing to?

```
typedef char * charp ;
const charp p ;
```

## Answer

*p* is constant.

---

## Question *14.21*

In the following code is *p2* an integer or an integer pointer?

```
typedef int * ptr ;
ptr p1, p2 ;
```

## Answer

Integer pointer