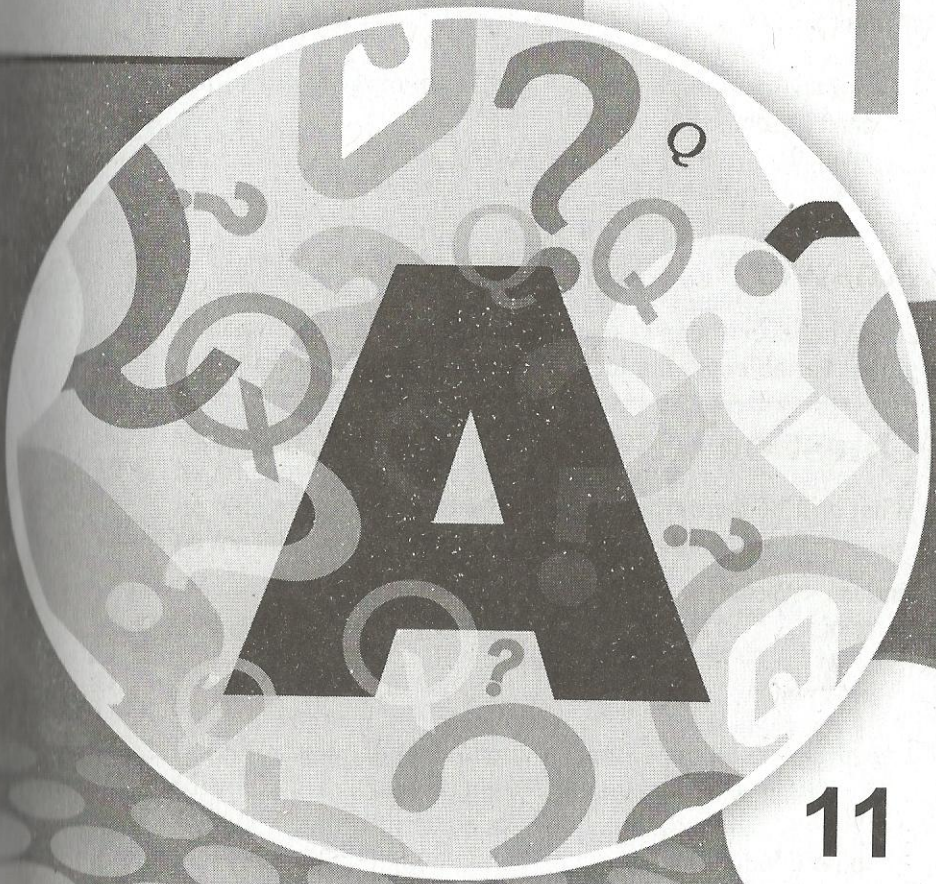
Declarations & Initializations

1



11

Question 1.1

What will be the output of the following program?

```
#include <stdio.h>
int x = 40 ;
int main( )
{
    int x = 20 ;
    printf ( "%d\n", x ) ;
    return 0 ;
}
```

Answer

20. Whenever there is a conflict between a local variable and a global variable it is the local variable that gets a priority.

Question 1.2

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    int x = 40 ;
    {
        int x  = 20 ;
        printf ( "%d ", x ) ;
    }
    printf ( "%d\n", x ) ;
    return 0 ;
}
```


Answer

20 40. In case of a conflict between two local variables the one that is more local gets the priority. More local means the one that is nearer to the point of usage. In this case the point of usage is the call to *printf( )* function.

Question 1.3

In the following program how long will the *for* loop get executed?

```
#include <stdio.h>
int main( )
{
    int i ;
    for ( ; scanf ( "%d", &i ) ; printf ( "%d\n", i ) ) ;
    return 0 ;
}
```

A. The *for* loop would not get executed at all.
B. The *for* loop would get executed only once.
C. The *for* loop would get executed 5 times.
D. The *for* loop would get executed infinite times.

Answer

D

Because return value of the function *scanf( )* would act as result of the condition. This value will always be non-zero, hence true.

Question 1.4

What do you mean by scope of a variable? What are the different types of scopes that a variable can have?

Answer

Scope indicates the region over which the variable's declaration has an effect. The four kinds of scopes are—file, function, block and prototype.

Question *1.5*

Which of the following statement is a declaration and which is a definition?

```
extern int i ;
int j ;
```

Answer

First is declaration, second is definition.

Question *1.6*

What are the differences between a declaration and a definition?

Answer

There are two differences between a declaration and a definition:

In the definition of a variable space is reserved for the variable and some initial value is given to it, whereas a declaration only identifies the type of the variable. Thus definition is the place where the variable is created or assigned storage, whereas declaration refers to places where the nature of the variable is stated but no storage is allocated.

Secondly, redefinition is an error, whereas, redeclaration is not an error.

Question *1.7*

Identify which of the following are declarations and which are definitions.

```
extern int x ;
float y ;
double pow ( double, double ) ;
float square ( float x ) { .. }
```

Answer

```
extern int x ;                  /* Declaration */
float y ;                       /* Definition */
double pow ( double, double ) ; /* Declaration */
float square ( float x ) { .. } /* Definition */
```

Question *1.8*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    extern int i ;
    i = 20 ;
    printf ( "%d\n", sizeof ( i ) ) ;
    return 0 ;
}
```

A. 2
B. 4
C. Would vary from compiler to compiler
D. Error, *i* undefined

Answer

D. *extern int i* is a declaration and not a definition, hence the error.

Question *1.9*

Is it true that a global variable may have several declarations, but only one definition? [Yes/No]

Answer

Yes

Question *1.10*

Is it true that a function may have several declarations, but only one definition? [Yes/No]

Answer

Yes

Question *1.11*

In the following program where is the variable *a* getting defined and where is it getting declared?

```
#include <stdio.h>
int main( )
{
    extern int a ;
    printf ( "%d\n", a ) ;
    return 0 ;
}
int a = 20 ;
```


Answer

extern int a is the declaration, whereas *int a = 20* is the definition.

Question *1.12*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    extern int a ;
    printf ( "%d\n", a ) ;
    return 0 ;
}
int a = 20 ;
```

A. 20
B. 0
C. Garbage value
D. Error

Answer

A

Question *1.13*

If the definition of an external variable occurs in the source file before its use in a particular function, then there is no need for an *extern* declaration in the function. [True/False]

Answer

True

Question 1.14

Which statement should you add to the function *main( )* to make it work?

```
#include <stdio.h>
int main( )
{
    printf ( "%d\n", z ) ;
    return 0 ;
}
int z = 25 ;
```

Answer

```
extern int z ;
```

Question 1.15

Suppose a program is divided into three files *f1*, *f2* and *f3*, and a variable is defined in the file *f1* but used in the files *f2* and *f3*. In such a case would we need the *extern* declaration for the variables in the files *f2* and *f3*? [Yes/No]

Answer

Yes

Question 1.16

When we mention the prototype of a function are we defining the function or declaring it?

Answer

We are declaring it. When the function, along with the statements belonging to it is mentioned, we are defining the function.

Question 1.17

What's the difference between the following declarations?

```
extern int fun( ) ;
int fun( ) ;
```

Answer

There is no difference except for the fact that the first one gives a hint that the function *fun( )* is probably in another source file.

Question 1.18

Why does the following program report an "display error on compilation?

```
#include <stdio.h>
int main( )
{
    display( ) ;
    return 0 ;
}
void display( )
{
    printf ( "Cliffhanger\n" ) ;
}
```

Answer

The error is reported because *display( )* is being called before it is defined. To avoid this error we must declare the prototype of *display( )* at the top.

Question *1.19*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    extern int fun ( float ) ;
    int a ;
    a = fun ( 3.14 ) ;
    printf ( "%d\n", a ) ;
    return 0 ;
}
int fun ( aa )
float aa ;
{
    return ( ( int ) aa ) ;
}
```

A. 3
B. 3.14
C. 0
D. Error

Answer

D. The error occurs because we have mixed the ANSI prototype with Kernighan & Ritchie style of function definition. When we use ANSI prototype for a function and pass a *float* to the function it is promoted to a *double*. When the function accepts this *double* into a *float* a type mismatch occurs hence the error. The remedy for this error could be to define the function as:

```
int fun ( float aa )
{
    ..
}
```


Question *1.20*

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    char *s1 ;
    char far *s2 ;
    char huge *s3 ;
    printf ( "%d %d %d\n", sizeof ( s1 ), sizeof ( s2 ), sizeof ( s3 ) ) ;
    return 0 ;
}
```

Answer

In a 16-bit compiler like Turbo C / C++ the output will be 4 4 2. However, in a 32-bit compiler like Visual Studio or gcc, the compiler will report an error since 32-bit compilers do not recognize *near*, *far* and *huge* pointers. In 32-bit compilers every pointer is 4 bytes wide.

Question *1.21*

Point out the error, if any, in the following program.

```
#include <stdio.h>
int main( )
{
    char *cptr, c ;
    void *vptr, v ;
    c = 10 ; v = 0 ;
    cptr = &c ;
    vptr = &v ;
    return 0 ;
}
```

Answer

Error: 'Size of 'v' is unknown or zero'. This is because *v* is declared as *void*. We can declare a *void* pointer but not a simple variable of type *void*.

Question *1.22*

Point out the error, if any, in the following program.

```
#include <stdio.h>
struct emp
{
    char name[20] ;
    int age ;
}
/* some more code may go here */
int fun ( int aa )
{
    int bb ;
    bb = aa * aa ;
    return ( bb ) ;
}
int main( )
{
```



```
    int a ;
    a = fun ( 20 ) ;
    printf ( "%d\n", a ) ;
    return 0 ;
}
```

Answer

Because of the missing semicolon at the end of the structure declaration (the intervening comment further obscures it) the compiler believes that *fun( )* would return something of the type *struct emp*, whereas, in actuality it is attempting to return an *int*. This causes a mismatch, resulting in an error.

Question *1.23*

Is the order of prototype declaration and structure declaration in the following program correct?

```
#include <stdio.h>
void f ( struct emp ) ;
struct emp
{
    char name[20] ;
    int age ;
} ;
int main( )
{
    struct emp e = { "Soicher", 34 } ;
    f ( e ) ;
    return 0 ;
}
void f ( struct emp ee )
{
    printf ( "%s %d\n", ee.name, ee.age ) ;
}
```

Answer

Yes. The order of declarations does not matter.

Question 1.24

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    struct emp
    {
        char name[20] ;
        int age ;
        float sal ;
    } ;
    struct emp e = { "Tiger" } ;
    printf ( "%d %f\n", e.age, e.sal ) ;
    return 0 ;
}
```

A. 0 0.000000
B. Garbage values
C. Error
D. None of the above

Answer

A. When an automatic structure is partially initialised, the remaining elements of the structure are initialised to 0.

Question 1.25

What will be the output of the code snippet given below?


```
#include <stdio.h>
int main( )
{
    struct book
    {
        char name[ 20 ] ;
        int noofpages ;
        float price ;
    } ;
    struct book b = { 0 } ;
    printf ( "%d %f\n", b.noofpages, b.price ) ;
    return 0 ;
}
```

A. 0 0.000000
B. Garbage values
C. Error
D. None of the above

Answer

A

Question 1.26

Is there any other way in which the same effect as in Question 1.25 can be obtained?

Answer

The same effect can be obtained using the following statement:

```
static struct book b ;
```

Being static, all elements of *b* would be initialized to 0.

Question 1.27

Some books suggest that the following definitions should be preceded by the word *static*. Is it correct?

```
int a[ ] = { 2, 3, 4, 12, 32 } ;
struct emp e = { "sandy", 23 } ;
```

Answer

Pre-ANSI C compilers had such a requirement. Compilers which conform to ANSI C standard do not have such a requirement.

Question 1.28

Point out the error, if any, in the following program.

```
#include <stdio.h>
int main( )
{
    union a
    {
        int i ;
        char ch[2] ;
    } ;
    union a z = { 512 } ;
    printf ( "%d %d", z.ch[0], z.ch[1] ) ;
    return 0 ;
}
```

Answer

A pre-ANSI C compiler will report an error 'Illegal structure operation'. In a pre-ANSI compiler a *union* variable cannot be initialised. However, ANSI C permits initialisation of first member of the *union*. In this case the output would be 0 2.

Question 1.29

If you are to share the variables or functions across several source files how would you ensure that all definitions and declarations are consistent?

Answer

The best arrangement is to place each definition in a relevant '.c' file. Then, put an external declaration in a header file ('.h' file) and use *#include* to bring in the declaration wherever needed. The '.c' file which contains the definition should also include the header file, so that the compiler can check that the definition matches the declaration.

Question 1.30

Global variables are available to all functions. Does there exist a mechanism by way of which I can make it available to some and not to others?

Answer

No. The only way this can be achieved is to define the variable locally in *main( )* instead of defining it globally and then passing it to the functions which need it.

Question 1.31

What do you mean by a translation unit?

Answer

A translation unit is a set of source files seen by the compiler and translated as a unit—generally one '.c' file, plus all header files mentioned in *#include* directives.

Question 1.32

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int a[5] = { 2, 3 } ;
    printf ( "%d %d %d\n", a[2], a[3], a[4] ) ;
    return 0 ;
}
```

A. Garbage values
B. 2 3 3
C. 3 2 2
D. 0 0 0

Answer

D. When an automatic array is partially initialized, the remaining array elements are initialized to 0.

Question 1.33

Point out the error, if any, in the following program.

```
#include <stdio.h>
int main( )
{
    int ( *p )( ) = fun ;
    ( *p )( ) ;
    return 0 ;
}
int fun( )
{
    printf ( "Loud and clear\n" ) ;
    return 0 ;
}
```


Answer

Here we are initializing the function pointer *p* to the address of the function *fun( )*. But during this initialization the function has not been defined. Hence an error. To eliminate this error add the prototype of the *fun( )* before declaration of *p*, as shown below:

```
extern int fun( ) ;
```

or simply

```
int fun( ) ;
```

Question 1.34

Which of the following set of statements is correct?

A.
```
typedef long a ;
extern int a c ;
```

B.
```
typedef long a
extern a int c ;
```

C.
```
typedef long a ;
extern a c ;
```

Answer

C

Question 1.35

What are *LG* and *LGP* in the following program?

```
int main( )
{
    typedef long LG, *LGP ;
    extern LGP lptr ;
    return 0 ;
}
```

Answer

Here, the first statement declares *LG* as a *typedef* for *long* and *LGP* as a pointer to a *long*. The second statement declares *lptr* of type *LGP* with storage class as *extern*.

Question *1.36*

Which of the following is correct about *err* used in the declaration given below?

```
typedef enum error { warning, test, exception } err ;
```

A. It is a *typedef* for *enum error*.
B. It is a variable of type *enum error*.
C. The statement is erroneous.
D. It is a structure.
E. It is one of the elements of *enum error*.

Answer

A

Question *1.37*

What are the different types of linkages?

Answer

There are three different types of linkages—external, internal and none. External linkage means global, non-static variables and functions, internal linkage means static variables and functions with file scope, and no linkage means local variables.

Question *1.38*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int x = 10, y = 20, z = 5, i ;
    i = x < y < z ;
    printf ( "%d\n", i ) ;
    return 0 ;
}
```

A. 0
B. 1
C. Error
D. None of the above

Answer

B. Since $x < y$ turns out to be true it is replaced by 1. This 1 is then compared with 5. Since this condition also turns out to be true it is replaced by 1. This 1 is then assigned to *i*.

Question *1.39*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    enum status { pass, fail, atkt } ;
    enum status stud1, stud2, stud3 ;
    stud1 = pass ;
    stud2 = atkt ;
    stud3 = fail ;
    printf ( "%d %d %d\n", stud1, stud2, stud3 ) ;
    return 0 ;
}
```

A. 0 1 2
B. 1 2 3
C. 0 2 1
D. 1 3 2
E. pass atkt fail

Answer

C. Enum elements always take values like 0, 1, 2, etc.

Question *1.40*

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    union a
    {
        short int i ;
        char ch[2] ;
    } ;
    union a u ;
    u.ch[0] = 3 ;
    u.ch[1] = 2 ;
    printf ( "%d %d %d\n", u.ch[0], u.ch[1], u.i ) ;
    return 0 ;
}
```


A. 3 2 515
B. 515 2 3
C. 3 2 5
D. None of the above

Answer

A

Question *1.41*

Consider the following code snippet:

```
struct data1
{
    int i ;
    float f ;
} ;

struct data2
{
    char name[10] ;
    double d ;
} ;
```

What is the correct way of declaring a *union info* which would share space amongst structure *data1* and *data2*?

Answer

```
union info
{
    struct data1  x ;
    struct data2  y ;
};
```

Question *1.42*

Which of the following structure declaration is incorrect?

A.
```
struct aa
{
    int a ;
    float b ;
};
```

B.
```
struct aa
{
    int a ;
    float b ;
    struct aa var ;
};
```

C.
```
struct aa
{
    int a ;
    float b ;
    struct aa *var ;
};
```

D.
```
struct aa
{
    int a ;
    float b ;
    struct aa **var ;
};
```


Answer

B

Question *1.43*

Which of the following special symbol is allowed in a variable name?

A. *
B. |
C. -
D. _

Answer

D

Question *1.44*

By default a real number is treated as a

A. float
B. double
C. long double
D. Depends on the compiler you are using

Answer

B

Question 1.45

Which of the following definition is correct?

A. int length ;
B. char int ;
C. int long ;
D. float double ;

Answer

A

Question 1.46

Which of the following structure declaration is correct?

A.
```
struct book
{
    char name[10] ;
    float price ;
    int pages ;
} ;
```

B.
```
struct
{
    char name[10] ;
    float price ;
    int pages ;
}
```

C.
```
struct book
{
    char name[10]
    float price
    int pages
} ;
```


D. All of the above.

Answer

A. In B semicolon is missing after the declaration. In C semicolon is missing after each structure element.

Question 1.47

Which of the following is not a user-defined data type?

A.
```
struct book
{
    char name[10] ;
    float price ;
    int pages ;
} ;
```

B.
```
long int l = 23.5 ;
```

C.
```
enum day { Sun, Mon, Tue, Wed } ;
```

D.
```
union a
{
    int i ;
    char ch[2] ;
} ;
```

Answer

B

Question 1.48

State True or False:

A. Sizes of *short integer* and *long integer* would vary from one platform to another.

B. Sizes of *short integer* and *long integer* can be verified using the *sizeof( )* operator.

C. A *float* is 4 bytes wide, whereas a *double* is 8 bytes wide.

D. Range of a float is -2.25e-308 to +2.25e+308.

E. Range of a double is -1.7e-38 to +1.7e38.

F. A *long double* should be used if range of a *double* is not enough to accommodate a real number.

G. The modulus operator cannot be used with a *long double*.

H. A char variable can store either an ASCII character or a Unicode character.

I. If *scanf( )* is used to store a value in a *char* variable then along with the value a carriage return (*\r*) also gets stored in it.

J. A *short* integer is at least 16 bits wide and a *long* integer is at least 32 bits wide.

K. By default, the data type of a constant without a decimal point is *int*, whereas the one with a decimal point is a *double*.

Answer

A. True
B. True
C. True
D. False
E. False
F. True
G. True
H. True
I. False
J. True
K. True

Question 1.49

Which of the following operations are INCORRECT?

A. `int i = 35 ; i = i % 5 ;`
B. `short int j = 255 ; j = j % 15 ;`
C. `long int k = 365L ; k = k % 24 ;`
D. `float a = 3.14 ; a = a % 3 ;`
E. `double d = 6.28 ; d = d % 4 ;`

Answer

D, E

Question 1.50

What statement will you add to the following code snippet if it is to produce the output given below?

```
31.240000 6.360000 5.460000
```

```
float a = 31.24;
double b = 6.36 ;
long double c = 5.46 ;
/* add statement here */
```

Answer

```
printf ( "%f %lf %Lf\n", a, b, c ) ;
```

Question *1.51*

Which of the following correctly represents a *long double* constant?

A. 6.68
B. 6.68L
C. 6.68f
D. 6.68lf
E. 6.68LF
F. 6.68LD
G. 6.68ld

Answer

B

Question *1.52*

Which of the following statement should be used to obtain a remainder after dividing 3.14 by 2.1?

A. `rem = 3.14 % 2.1 ;`
B. `rem = modf ( 3.14, 2.1 ) ;`
C. `rem = fmod ( 3.14, 2.1 ) ;`
D. `rem = 3.14 mod 2.1`

E. We cannot obtain the remainder in floating point division

Answer

E

Question *1.53*

How would you round off a value 1.66 to 2.000000 and truncate 1.75 to 1.000000?

Answer

```
#include <math.h>
int main( )
{
    float x = 1.66 ;
    float y = 1.75 ;
    printf ( "%f %f\n", ceil ( x ), floor ( y ) ) ;
    return 0 ;
}
```

Question *1.54*

What is *size_t* ?

Answer

It is the type of the result of the *sizeof* operator. *size_t* is used to express the size of something or the number of characters in something. For example, it is the type that you pass to *malloc( )* to indicate how many bytes you wish to allocate. Or it is the type returned by *strlen( )* to indicate the number of characters in a string.

Each compiler implementation chooses a type like *unsigned int* or *unsigned long* (or something else) to be its *size_t*, depending on what makes most sense. Each implementation publishes its own choice of *size_t* in several header files like 'stdio.h', 'stdlib.h', etc. In most implementations *size_t* is defined as:

```
typedef unsigned int size_t ;
```

This means that on this particular implementation *size_t* is an *unsigned int*. Other implementations may make other choices.

What is important is that you should not worry about what *size_t* looks like for a particular implementation; all you should care about is that it is the *right* type for representing object sizes and count.

Question *1.55*

What is the storage class of *i* in the following statement?

```
int i ;
```

Answer

The storage class of *i* is *auto* if it is declared within *main( )* and *extern* if it is declared outside *main( )*.

Question *1.56*

Can the variables *i* and *j* declared below be used in any other file of the same program?

```
auto int i ;
static int j ;
int main( )
```



```
{
    ...
    return 0 ;
}
```

Answer

Variable *i* can be used provided it is declared as

```
extern int i ;
```

in the file in which it is being used. *j* being static cannot be used in other files.

Question *1.57*

What values would be stored in variables *i* and *j* defined below?

```
int i ;
float j ;
```

Answer

The values stored in these variables depend upon where they are defined. If they are defined outside all functions, they would be treated as external storage class variables and hence would contain default values 0 and 0.0 respectively.

However, if they are defined inside a function then they would be treated as automatic storage class variables. In this case the variables would contain garbage values.

Question *1.58*

Is it possible that the number of bytes occupied by a *short int* is more than that occupied by an *int*?

Answer

No

Question *1.59*

Is it possible that the number of bytes occupied by an *int* is more than that occupied by a *long int*?

Answer

No