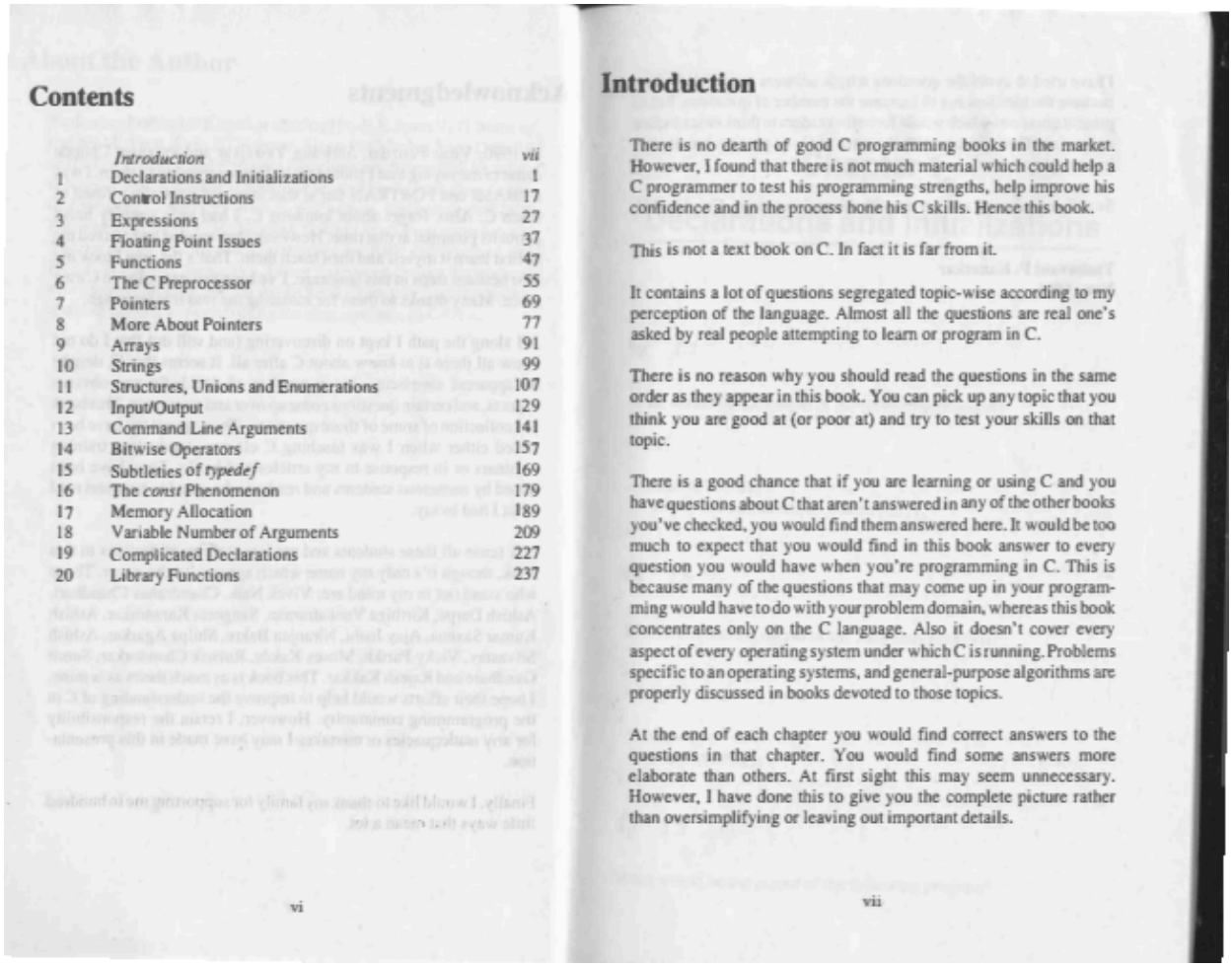


Contents

Introduction

There is no dearth of good C programming books in the market. However, I found that there is not much material which could help a C programmer to test his programming strengths, help improve his confidence and in the process hone his C skills. Hence this book.

This is not a text book on C. In fact it is far from it.

It contains a lot of questions segregated topic-wise according to my perception of the language. Almost all the questions are real one's asked by real people attempting to learn or program in C.

There is no reason why you should read the questions in the same order as they appear in this book. You can pick up any topic that you think you are good at (or poor at) and try to test your skills on that topic.

There is a good chance that if you are learning or using C and you have questions about C that aren't answered in any of the other books you've checked, you would find them answered here. It would be too much to expect that you would find in this book answer to every question you would have when you're programming in C. This is because many of the questions that may come up in your programming would have to do with your problem domain, whereas this book concentrates only on the C language. Also it doesn't cover every aspect of every operating system under which C is running. Problems specific to an operating systems, and general-purpose algorithms are properly discussed in books devoted to those topics.

At the end of each chapter you would find correct answers to the questions in that chapter. You would find some answers more elaborate than others. At first sight this may seem unnecessary. However, I have done this to give you the complete picture rather than oversimplifying or leaving out important details.

Contents

Introduction

There is no dearth of good C programming books in the market. However, I found that there is not much material which could help a

C programmer to test his programming strengths, help improve his confidence and in the process hone his C skills. Hence this book.

This is not a text book on C. In fact it is far from it.

It contains a lot of questions segregated topic-wise according to my perception of the language. Almost all the questions are real one's asked by real people attempting to learn or program in C.

There is no reason why you should read the questions in the same order as they appear in this book. You can pick up any topic that you think you are good at (or poor at) and try to test your skills on that

topic.

There is a good chance that if you are learning or using C and you have questions about C that aren't answered in any of the other books you've checked, you would find them answered here. It would be too. much to expect that you would find in this book answer to every question b

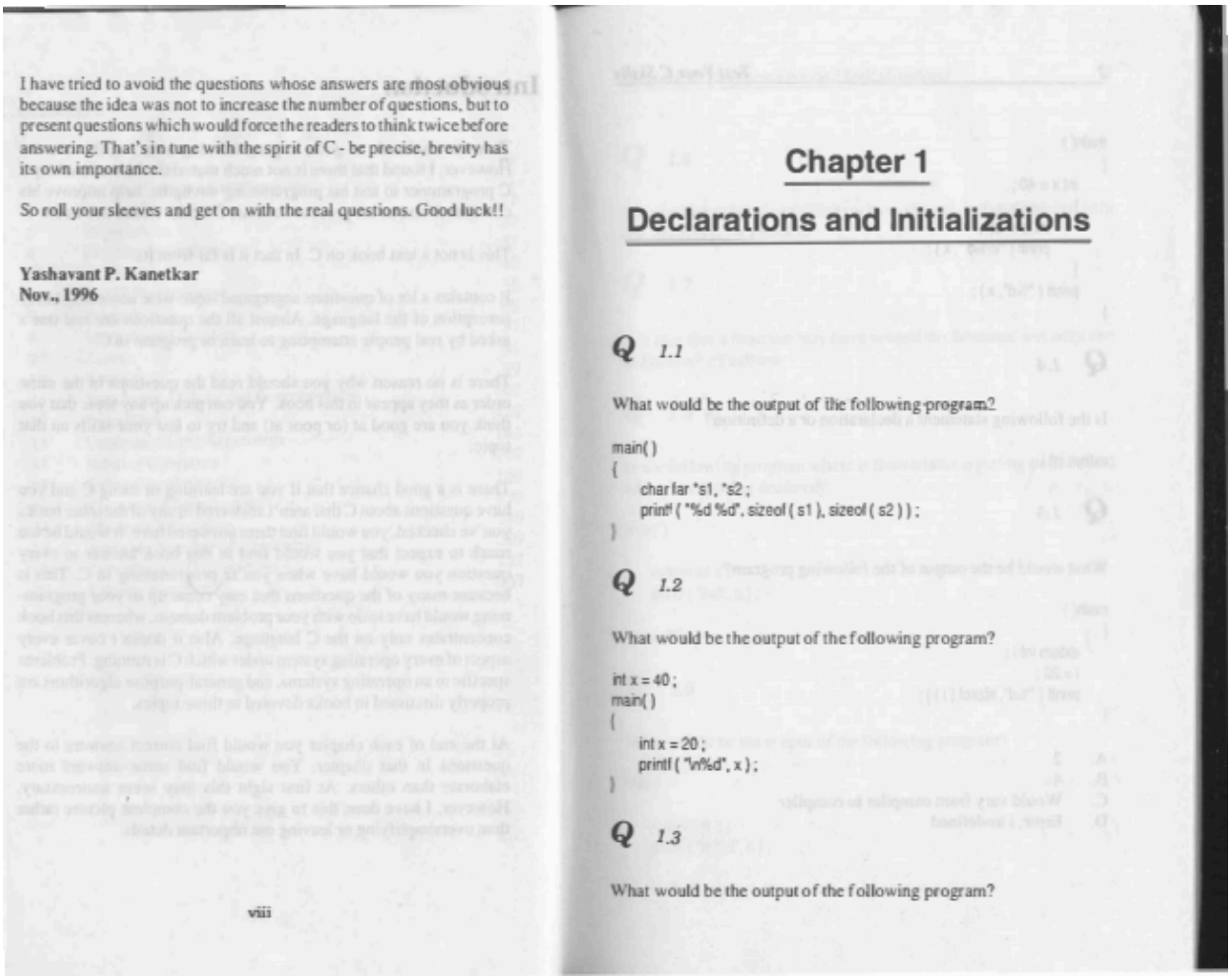
cause many you would of the have questions when that you're may programming come up in your in C. program This is

mmg would have to do with your problem domain, whereas this book concentrates only on the C language. Also it doesn't cover every

aspect of every operating system under which C is running. Problems specific to an operating systems, and general-purpose algorithms are

properly discussed in books devoted to those topics.

At the end of each chapter you would find correct answers to the questions in that chapter. You would find some answers more elaborate than others. At first sight this may seem unnecessary. However, I have done this to give you the complete picture rather than oversimplifying or leaving out important details.



I have tried to avoid the questions whose answers are most obvious because the idea was not to increase the number of questions, but to present questions which would force the readers to think twice before answering. That's in tune with the spirit of C - be precise, brevity has its own importance.

So roll your sleeves and get on with the real questions. Good luck!!

Yashavant P. Kanetkar
Nov., 1996

viii

Chapter 1

Declarations and Initializations

Q 1.1

What would be the output of the following program?

```
main( )
{
    char far *s1, *s2 ;
    printf ( "%d %d", sizeof ( s1 ), sizeof ( s2 ) ) ;
}
```

Q 1.2

What would be the output of the following program?

```
int x = 40 ;
main( )
{
    int x = 20 ;
    printf ( "\n%d", x ) ;
}
```

Q 1.3

What would be the output of the following program?

I have tried to avoid the questions whose answers are most obvious because the idea was not to increase the number of questions, but to present questions which would force the readers to think twice before answering. That's in tune with the spirit of C- be precise, brevity has its own importance.

So roll your sleeves and get on with the real questions. Good luck!!

Yashavant P. Kanetkar Nov., 1996

viii

-

Chapter 1

Declarations and lnitializations

Q

1.1

What would be the output offne·fo1Iuwing-progmm1.

```
main() {
char far *s1, *s2; pr
intf ( "%d %d", sizeof ( s1 ), sizeof ( s2)) ;
```

Q

1.2

What would be the output of the following program?

```
int
x =
        40; main() {
int
x =
                                                    20; printf ( "\n%d",
x
) ;
```

Q

1.3

What would be the output of the following program?

```
main( )
{
    int x = 40 ;
    {
        int x = 20 ;
        printf ( "\n%d ", x ) ;
    }
    printf ( "%d", x ) ;
}
```

Q *1.4*

Is the following statement a declaration or a definition?

```
extern int i ;
```

Q *1.5*

What would be the output of the following program?

```
main( )
{
    extern int i ;
    i = 20 ;
    printf ( "%d", sizeof ( i ) ) ;
}
```

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

Q *1.6*

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

Q *1.7*

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

Q *1.8*

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

```
main( )
{
    extern int a ;
    printf ( "%d", a ) ;
}
int a = 20 ;
```

Q *1.9*

What would be the output of the following program?

```
main( )
{
    extern int a ;
    printf ( "\n%d", a ) ;
}
int a = 20 ;
```

. 2

Test Your C Skills

main() {

int x

=

40; {

int x

=

20; printf ("\n%d

·,

x) ; } printf ('%d", x) ;

Q

1.4

Is the following statement a declaration or a definition?

ext ern int i ;

Q

1.5

What would be the output of the following program?

main( ) {

ext ern int i ; i

=

20 ; printf ("%d', sizeof (i))

;

A.

2 B. c.

4

I Would vary from compiler to compiler D.

Error, i undefined

Q

1.6

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

Q

1.7

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

Q

1.8

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

main() {

}

ext ern int a; printf ('%d', a) ;

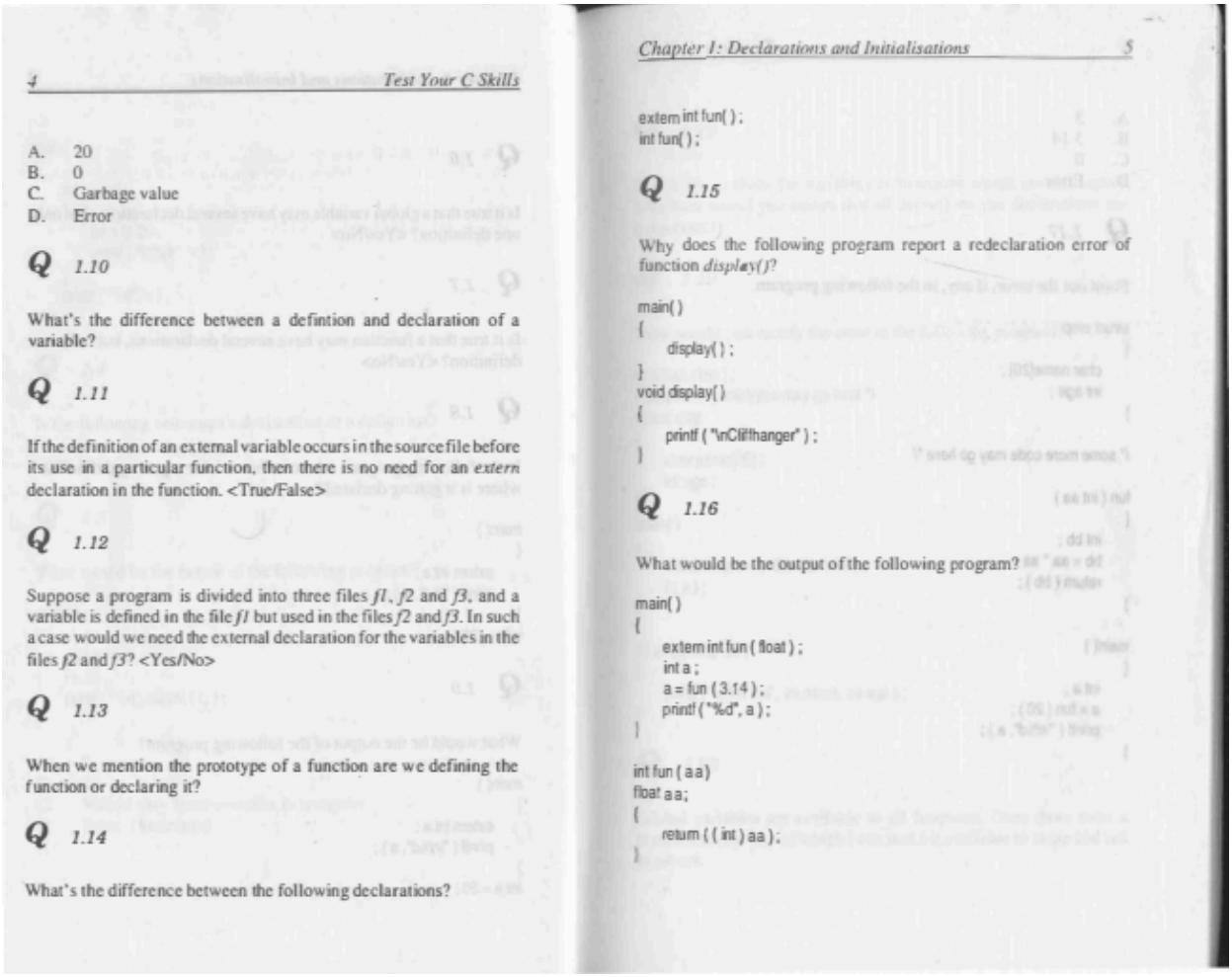
int a = 20 ;

Q

1.9

What would be the output of the following program?

```
main( ) {
externint a; printf ( "\n%d', a);
int a = 20;
```



4 Test Your C Skills

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

Q *1.10*

What's the difference between a defintion and declaration of a variable?

Q *1.11*

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>

Q *1.12*

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 external declaration for the variables in the files *f2* and *f3*? <Yes/No>

Q *1.13*

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

Q *1.14*

What's the difference between the following declarations?

Chapter 1: Declarations and Initialisations 5

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

Q *1.15*

Why does the following program report a redeclaration error of function *display( )*?

```
main( )
{
    display( ) ;
}
void display( )
{
    printf ( "\nCliffhanger" ) ;
}
```

Q *1.16*

What would be the output of the following program?

```
main( )
{
    extern int fun ( float ) ;
    int a ;
    a = fun ( 3.14 ) ;
    printf ( "%d", a ) ;
}

int fun ( aa )
float aa ;
{
    return ( ( int ) aa ) ;
}
```

Test 4

Your C Skills

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

Q

1.10

What's the difference between a defintion and declaration of a variable?

Q

1.11

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

Q

1.12

Suppose a program is divided into three files fl , j2 and f3, and a variable is defined in the filefl but used in the filesj2 andf3. In such a case would we need the external declaration for the variables in the filesj2 andf3? <Yes/No>

Q

1.13

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

.

Q

1.14

What's the difference between the following declarations?

-

extern in! f un( ) ; in! fun( ) ;

Q

1.15

5

Why does the following program report a rcdeclaration error of function display{)?

```
mai n( ) {
      display() ; } void display( ) {
pri ntf ( "\nCiiffhanger" ) ;
```

Q

1.16

What would be the output of the following program?

```
main( ) {
extern int fun ( float) ; i n! a ; a
=
     f un ( 3.14 ) ; printf ( "%d", a ) ;
int fun ( aa) float aa; {
r
etum ( ( int ) aa ) ;
```

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

Q 1.17

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

```
struct emp
{
    char name[20] ;
    int age ;
}

/* some more code may go here */

fun ( int aa )
{
    int bb ;
    bb = aa * aa ;
    return ( bb ) ;
}

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


Q 1.18

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

Q 1.19

How would you rectify the error in the following program?

```
f ( struct emp ) ;
/* any other prototypes may go here */
struct emp
{
    char name[20] ;
    int age ;
} ;
main( )
{
    struct emp e = { "Soicher", 34 } ;
    f ( e ) ;
}

f ( struct emp ee )
{
    printf ( "\n%s %d", ee.name, ee.age ) ;
}
```

Q 1.20

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.

---

6 Test Your C Skills

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

Q

1.17

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

struct emp {

char name(20) ; int age;

r some more code may go here */

fun

(intaa) {

1

intbb; bb=aa•aa; retum

(

bb);

main() {

inta; a= fun (20); printf ("\n%d"

I

a) ;

,

Q

1.18

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

consistent?

Q

1.19

How would you rectify the error in the following program?

f (struct emp

)

; r a

n

y other prototypes may go here *I struct emp {

char

name{20] ; int age; } : main() {

struct emp e

=

{ "Soicher"

I

34 } ;

t

(e);

f

(

struct emp ee

) {

printf

(

"\rflos %d", ee.namel ee.age) ;

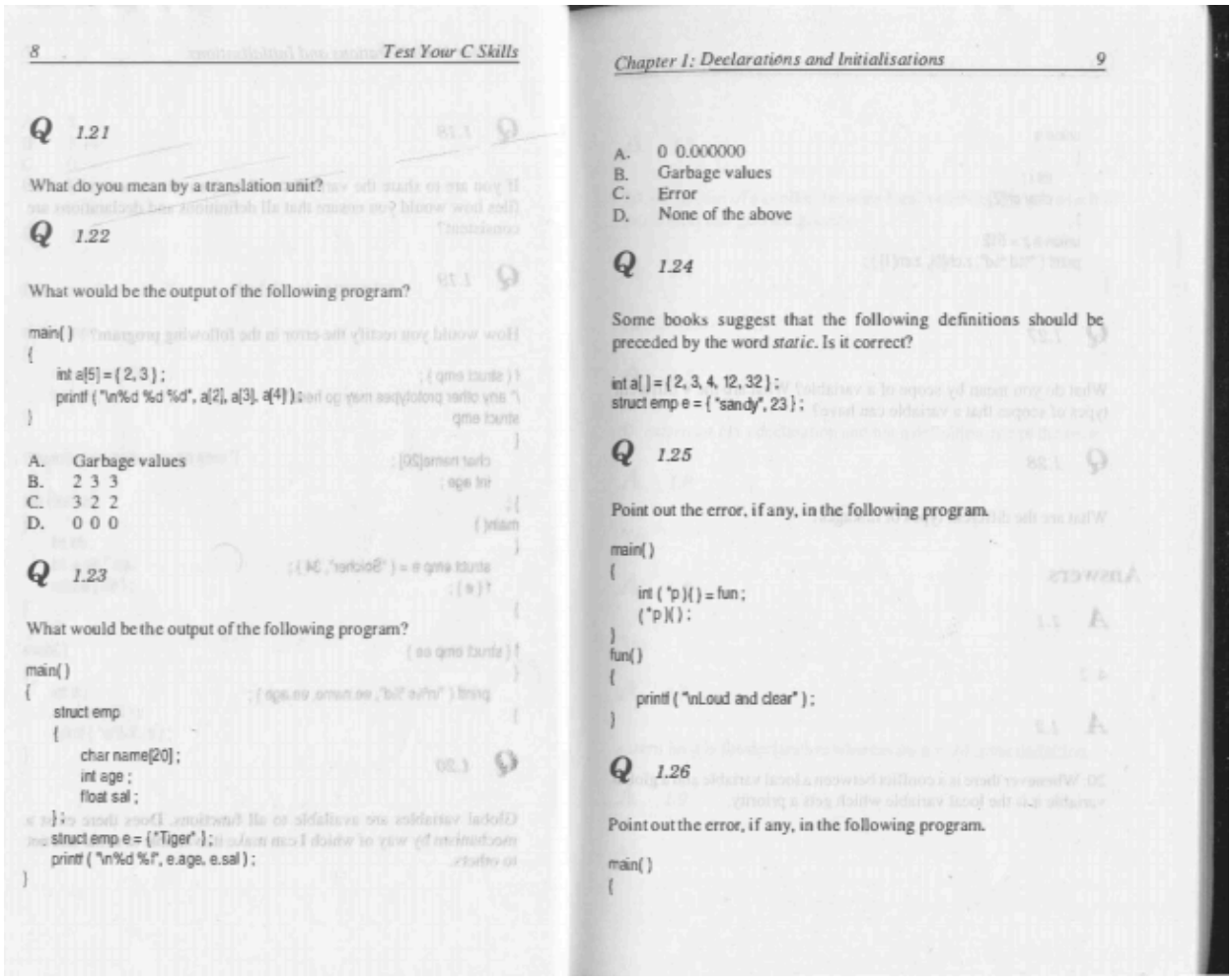
Q

1.20

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.



8 Test Your C Skills

Q 1.21

What do you mean by a translation unit?

Q 1.22

What would be the output of the following program?

```
main( )
{
    int a[5] = { 2, 3 } ;
    printf ( "\n%d %d %d", a[2], a[3], a[4] ) ;
}
```

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

Q 1.23

What would be the output of the following program?

```
main( )
{
    struct emp
    {
        char name[20] ;
        int age ;
        float sal ;
    } ;
    struct emp e = { "Tiger" } ;
    printf ( "\n%d %f", e.age, e.sal ) ;
}
```

Chapter 1: Declarations and Initialisations 9

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

Q 1.24

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 } ;
```

Q 1.25

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

```
main( )
{
    int ( *p )( ) = fun ;
    ( *p )( ) ;
}
fun( )
{
    printf ( "\nLoud and clear" ) ;
}
```

Q 1.26

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

```
main( )
{
```

---

8 Test Your C Skills

Q

1.21

What do you mean by a translation unit?

Q

1.22

What would be the output of the following program'>

main( ) {

int a[S]

=

{ 2, 3 } ; printf ("\n%d %d %d", a[2], a[3], a[4)) ;

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

Q

1.23

What would be the output of the following program?

```
main( ) {
struct emp {
} ;
char name[20] ; int age; float sal ;
. struct emp e
=
                                    { "Tiger" } ; prin tf ( "\n%d %f", e.age, e.sal);
```


Declaratitms and lnitialisations

9

A.

0 0.0000 B. Garbage values c.

Error D. None of the above

Q

1.24

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 } ;
```

Q

1.25

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

```
main() {
int ( *p )( )
=
            fun ; ( *p )( )
```

; } fun() {

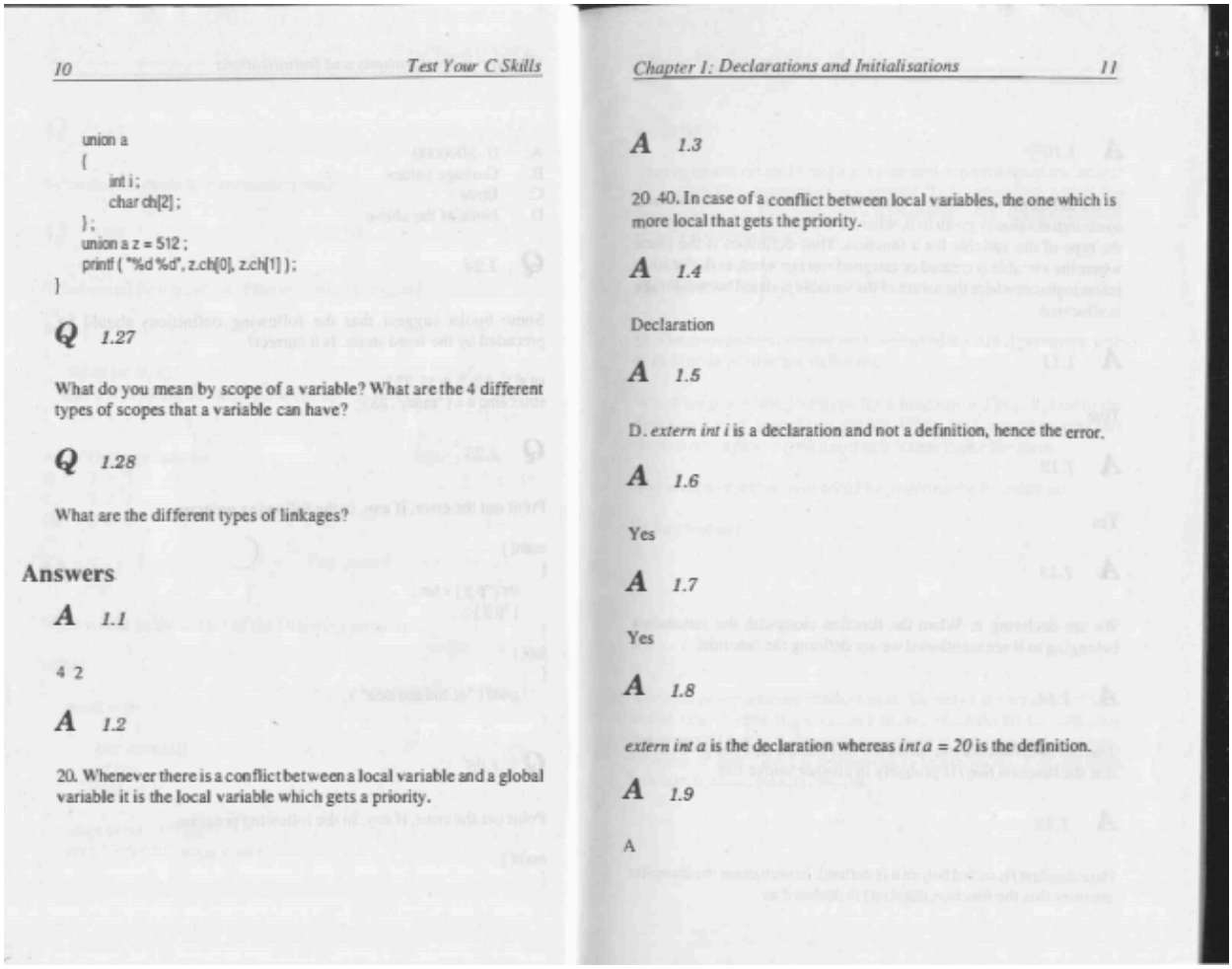
printf ("\nloud and clear") ;

Q

1.26

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

main( ) {



10 Test Your C Skills

```
union a
{
    int i;
    char ch[2];
};
union a z = 512;
printf ( "%d %d", z.ch[0], z.ch[1] );
}
```

Q *1.27*

What do you mean by scope of a variable? What are the 4 different types of scopes that a variable can have?

Q *1.28*

What are the different types of linkages?

Answers

A *1.1*

4 2

A *1.2*

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

Chapter 1: Declarations and Initialisations 11

A *1.3*

20 40. In case of a conflict between local variables, the one which is more local that gets the priority.

A *1.4*

Declaration

A *1.5*

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

A *1.6*

Yes

A *1.7*

Yes

A *1.8*

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

A *1.9*

A

Test Your C Skills

union a {

inti; char ch[2]; } ; union a z

=

10

512; printf ("%d %d", z.ch[O], z.ch[1]);

Q

1.27

What do you mean by scope of a variable? What are the 4 different types of scopes that a variable can have?

Q

1.28

What are the different types of linkages?

Answers.

A

1.1

4 2

A

1.2

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

1: Chapter

Declarations and Initialisations 11

A

1.3

20 40. In case of a conflict between local variables, the one which is more local that gets the priority.

A

1.4

Declaration

A

1.s

D. extern inti is a declaration and not a definition, hence the error.

A

1.6

Yes

A

1.7

Yes

A

1.8

extern int a is the declaration whereas int a

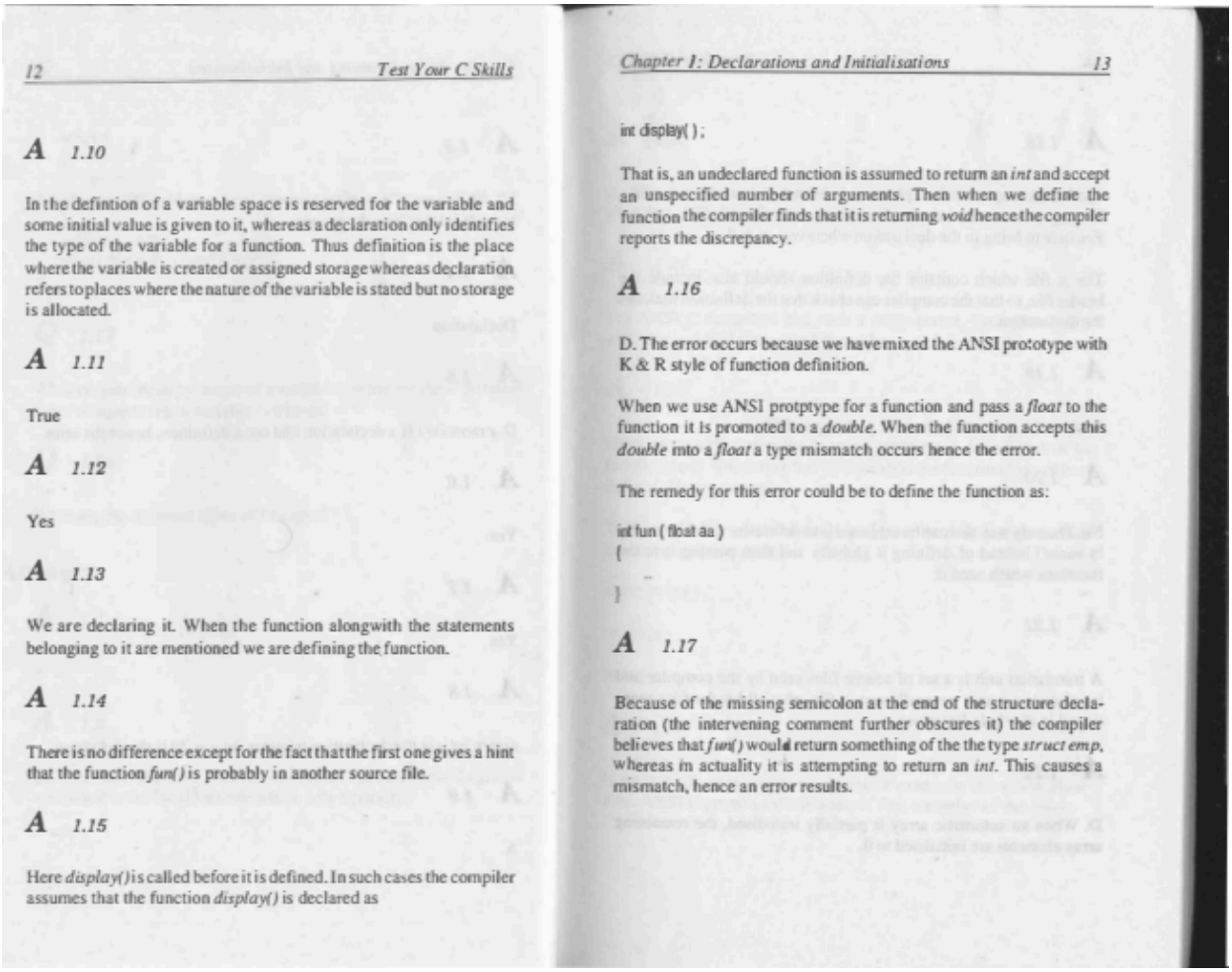
=

20 is the definition.

A

1.9

A



12 Test Your C Skills

A

1.10

In the defintion 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 for a function. 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.

A

1.11

True

A

1.12

Yes

A

1.13

We are declaring it. When the function alongwith the statements belonging to it are mentioned we are defining the. function.

A

1.14

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.

A

1.1s

Here display() is called before it is defined. In such cases the compiler assumes that the function display() is declared as

Chapter

1:

Declarations and Initialisations 13

int display( ) ;

That is, an undeclared function is assumed to return an int and accept an unspecified number of arguments. Then when we define the function the compiler finds that it is returning void hence the compiler reports the discrepancy.

A

1.16

D. The error occurs because we have mixed the ANSI pro�otype with K & R style of function definition.

Whe� funct10� w� It

u.se IS

promoted ANSI protptype to a double. for a When function the and function pass afl

accepts o

at to this the

double mto afl

o

at a type mismatch occurs hence the error.

The remedy for this error could be to define the function as:

int fun (float aa) {

A

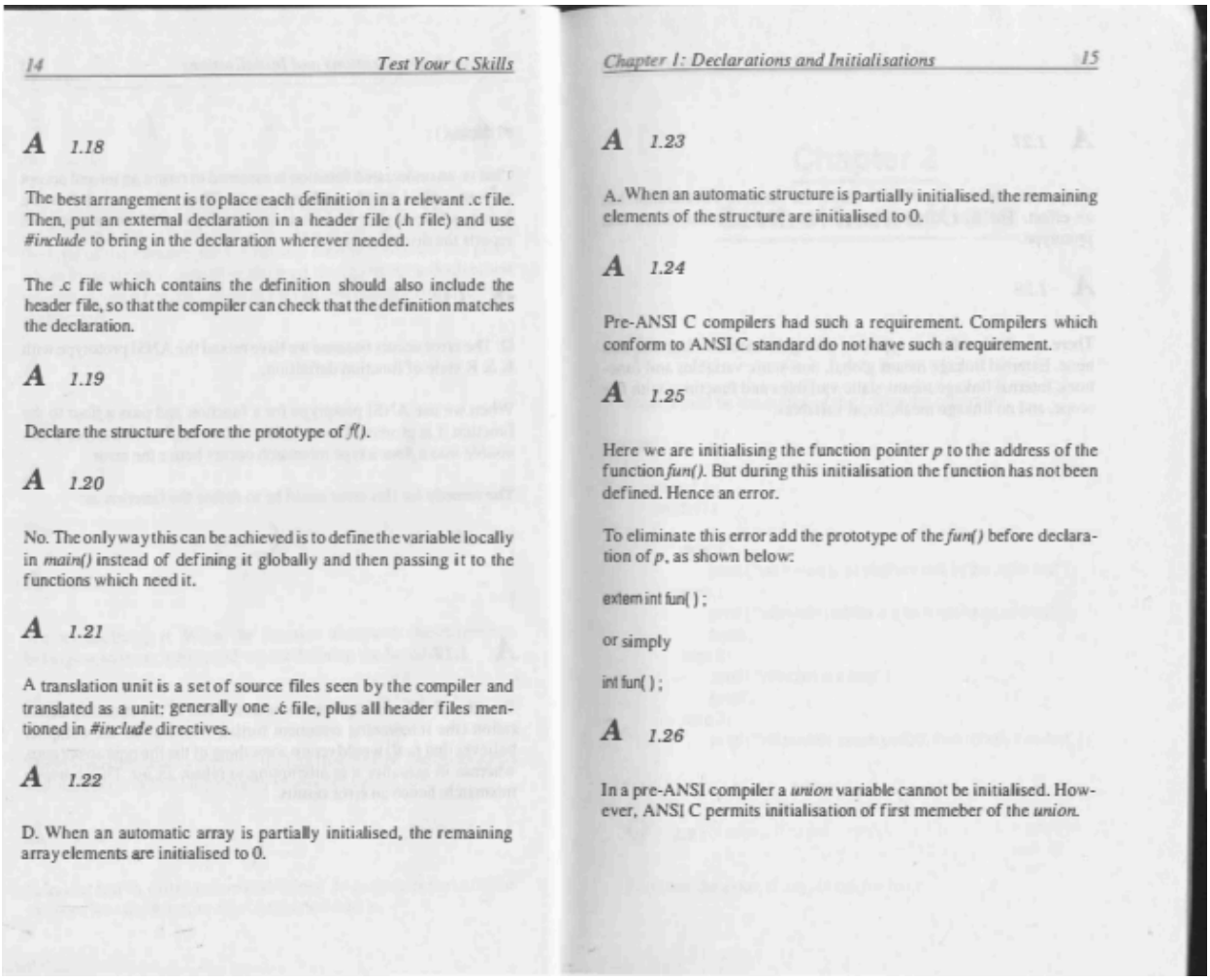
1.17

Because of the missing semicolon at the end of the structure decla

w belt ration &#65533;

ereas eves (the &#65533;hat m intervening fun(J_wo&#65533;l

? comment return something further obscures of the the it) the compiler type struct emp, actuality 1F1s attempting to return an int. This causes a mismatch, hence an error results.



14 Test Your C Skills

A 1.18

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.

A 1.19

Declare the structure before the prototype of *f()*.

A 1.20

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.

A 1.21

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.

A 1.22

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

Chapter 1: Declarations and Initialisations 15

A 1.23

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

A 1.24

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

A 1.25

Here we are initialising the function pointer *p* to the address of the function *fun()*. But during this initialisation 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( ) ;
```

A 1.26

In a pre-ANSI compiler a *union* variable cannot be initialised. However, ANSI C permits initialisation of first memeber of the *union*.

---

14 Test Your C Skills

A

1.1s

The best arrangement is to place each aefinition 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.

A

1.19

Declare the structure before the prototype off().

A

1.2o

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.

A

1.21

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 men tioned in #include directives.

A

1.22

D. When an automatic array is partially initialised, the remaining array elements

are

initialised to 0.

•

Chapter

I:

15

A

1.23

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

A

1.24

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

A

1.2s

Here we are initialising the function pointer p to the address of the function fun(). But during this initialisation the function has not been defined. Hence an error.

To eliminate this error add the prototype of the fun() before declara tion of

p,

Declarations and Initialisations

as shown below:

extern int fun( ) ;

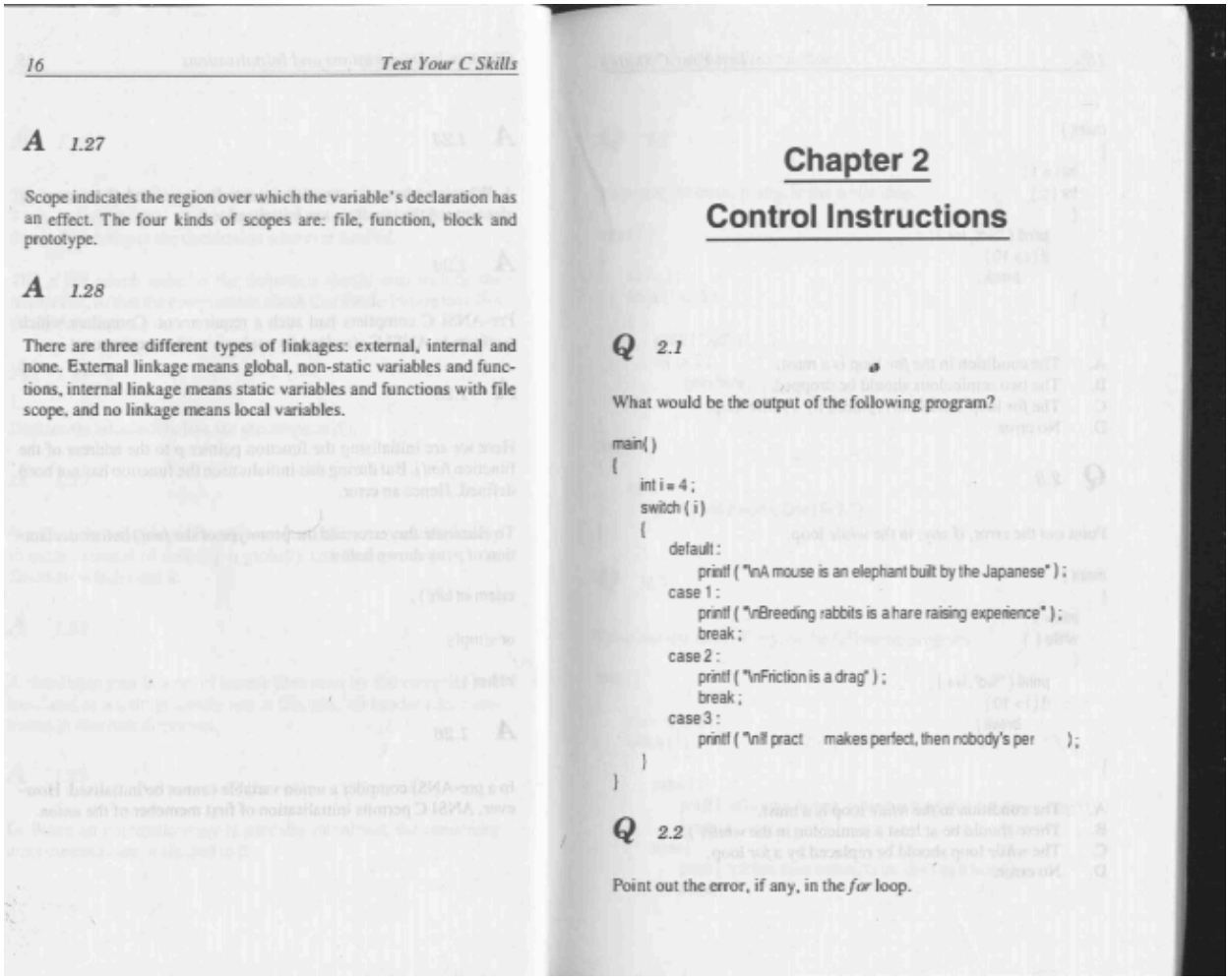
or simply

int fun( ) ;

A

1.26

In a pre-ANSI compiler a union variable cannot be initialised. How ever, ANSI C permits initialisation. of first memeber of the union.



16 Test Your C Skills

A *1.27*

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

A *1.28*

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.

Chapter 2

Control Instructions

Q *2.1*

What would be the output of the following program?

```
main( )
{
    int i = 4 ;
    switch ( i )
    {
        default :
            printf ( "\nA mouse is an elephant built by the Japanese" ) ;
        case 1 :
            printf ( "\nBreeding rabbits is a hare raising experience" ) ;
            break ;
        case 2 :
            printf ( "\nFriction is a drag" ) ;
            break ;
        case 3 :
            printf ( "\nIf pract     makes perfect, then nobody's per      ) ;
    }
}
```

Q *2.2*

Point out the error, if any, in the *for* loop.

·

16 Test Your C Skills

A

1.21

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

A

1.2s

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

Chapter 2

Control Instructions

Q

2.1

..

.,

What would be the output of the following program?

```
main( ) {
int i = 4 ; switch ( i) {
default :
      printf ( "\nA mouse is an elephant built by the Japanese" ) ; case 1 :
pr intf ( "\nBreeding rabbits is a hare raising experience • ) ; break ; case 2:
pr intf ( "\nFriction is a drag" ) ; break ; case 3:
printf ( "\n lf prac tice makes perfect , then nobody's pe rfect" ) ;
```

Q

2.2

Point out the error, if any, in the for loop.