Variable Number of Arguments

17

439

Question *17.1*

Which header file should you include if you are to develop a function, which can accept variable number of arguments?

A. vararg.h
B. stdlib.h
C. stdio.h
D. stdarg.h

Answer

D

Question *17.2*

What will be the output of the following program?

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

void fun ( char *msg, ... ) ;
int main( )
{
    fun ( "Nothing specific\n", 1, 4, 7, 11, 0 ) ;
    return 0 ;
}
void fun ( char *msg, ... )
{
    int tot = 0 ;
    va_list ptr ;
    int num ;
    va_start ( ptr, msg ) ;
    num = va_arg ( ptr, int ) ;
    num = va_arg ( ptr, int ) ;
    printf ( "%d\n", num ) ;
}
```


Answer

4

Question *17.3*

Is it necessary that in a function which accepts variable argument list there should at least be one fixed argument? [Yes/No]

Answer

Yes

Question *17.4*

Can the fixed arguments passed to the function that accepts variable argument list, occur at the end? [Yes/No]

Answer

No

Question *17.5*

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

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

void varfun ( int n, ... ) ;
int main( )
{
    varfun ( 3, 7, -11, 0 ) ;
    return 0 ;
}
void varfun ( int n, ... )
{
```

```
    va_list ptr ;
    int num ;
    num = va_arg ( ptr, int ) ;
    printf ( "%d\n", num ) ;
}
```

Answer

Since *ptr* has not been set at the beginning of the variable argument list using *va_start* it may be pointing anywhere and hence a call to *va_arg* would cause an Exception at runtime.

Question *17.6*

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

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

void varfun ( int n, ... ) ;
int main( )
{
    varfun ( 3, 7.5, -11.2, 0.66 ) ;
    return 0 ;
}
void varfun ( int n, ... )
{
    float *ptr ;
    int num ;
    va_start ( ptr, n ) ;
    num = va_arg ( ptr, int ) ;
    printf ( "%d\n", num ) ;
}
```


Answer

Irrespective of the type of argument passed to a function receiving variable argument list, *ptr* must be of the type *va_list*.

Question *17.7*

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

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

void fun ( ... ) ;
int main( )
{
    fun ( 1, 4, 7, 11, 0 ) ;
    return 0 ;
}
void fun ( ... )
{
    va_list ptr ;
    int num ;
    va_start ( ptr, int ) ;
    num = va_arg ( ptr, int ) ;
    printf ( "%d\n", num ) ;
}
```

Answer

There is no fixed argument in the definition of *fun( )*.

Question *17.8*

The macro *va_start* is used to initialize a pointer to the beginning of the list of fixed arguments. [True/False]

Answer

False

Question 17.9

The macro *va_arg* is used to extract an argument from the variable argument list and advance the pointer to the next argument. [True/False]

Answer

True

Question 17.10

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

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

void display ( int num, ... ) ;
int main( )
{
    display ( 4, 'A', 'a', 'b', 'c' ) ;
    return 0 ;
}
void display ( int num, ... )
{
    char c ; int j ;
    va_list ptr ;
    va_start ( ptr, num ) ;
    for ( j = 1 ; j <= num ; j++ )
    {
        c = va_arg ( ptr, char ) ;
        printf ( "%c ", c ) ;
    }
    printf ( "\n" ) ;
}
```


Answer

No error. It will output A a b c.

Question 17.11

What will be the output of the following program?

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

void display ( int num, ... ) ;
int main( )
{
    display ( 4, 'A', 'B', 'C', 'D' ) ;
    return 0 ;
}
void display ( int num, ... )
{
    char c, c1 ; int j ;
    va_list ptr ;
    va_list ptr1 ;
    va_start ( ptr, num ) ;
    va_start ( ptr1, num ) ;
    for ( j = 1 ; j <= num ; j++ )
    {
        c = va_arg ( ptr, int ) ;
        printf ( "%c, ", c ) ;
        c1 = va_arg ( ptr1, char ) ;
        printf ( "%d\n", c1 ) ;
    }
    printf ( "\n" ) ;
}
```

Answer

A, 65
B, 66
C, 67
D, 68

Question *17.12*

What will be the output of the following program?

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

void fun1 ( int num, ... ) ;
void fun2 ( int num; ... ) ;
int main( )
{
    fun1( 1, "Apple", "Boys", "Cats", "Dogs" ) ;
    fun2 ( 2, 12, 13, 14 ) ;
    return 0 ;
}
void fun1 ( int num, ... )
{
    va_list ptr ;
    char *str ;
    va_start ( ptr, num ) ;
    str = va_arg ( ptr, char * ) ;
    printf ( "%s ", str ) ;
}
void fun2 ( int num, ... )
{
    va_list ptr ;
    va_start ( ptr, num ) ;
    num = va_arg ( ptr, int ) ;
    printf ( "%d \n", num ) ;
```



```
}
```

Answer

Apple 12

Question *17.13*

Mention any variable argument-list function that you have used and its prototype.

Answer

```
int printf ( const char *format, ... ) ;
```

Question *17.14*

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

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

int main( )
{
    int display ( int num, ... ) ;
    display ( 4, 12.5, 13.5, 14.5, 44.3 ) ;
    return 0 ;
}
int display ( int num, ... )
{
    float c ; int j ;
    va_list ptr ;
    va_start ( ptr, num ) ;
    for ( j = 1 ; j <= num ; j++ )
    {
        c = va_arg ( ptr, float ) ;
        printf ( "%f \n", c ) ;
```

```
    }
    return 0 ;
}
```

Answer

We should use *double* in place of *float* as shown below:

```
int display ( int num, ... )
{
    double c ; int j ;
    va_list ptr ;
    va_start ( ptr, num ) ;
    for ( j = 1 ; j <= num ; j++ )
    {
        c = va_arg ( ptr, double ) ;
        printf ( "%lf \n", c ) ;
    }
    return 0 ;
}
```

Question *17.15*

State whether True or False.

va_list is an array that holds information needed by *va_arg* and *va_end*.

Answer

True

Question *17.16*

How can *%f* be used for both *float* and *double* arguments in *printf( )* ?

Answer

In variable length arguments lists, types *char* & *short int* are promoted to *int* and *float* is promoted to *double*.

Question *17.17*

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

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

int main( )
{
    void display ( char *s, int num1, int num2, ... ) ;
    display ( "Hello", 4, 2, 12.5, 13.5, 14.5, 44.0 ) ;
    return 0 ;
}
void display ( char *s, int num1, int num2, ... )
{
    double c ;
    va_list ptr ;
    va_start ( ptr, s ) ;
    c = va_arg ( ptr, double ) ;
    printf ( "%lf \n", c ) ;
}
```

Answer

While setting the *ptr* to the beginning of variable argument list we should have used *va_start (ptr, num2)*.

Question *17.18*

Can we pass a variable argument list to a function at run-time? [Yes/No]

Answer

No. Every actual argument list must be completely known at compile time. In that sense it is not truly a variable argument list.

Question *17.19*

While defining a variable argument list function can we drop the ellipsis (...) ? [Yes/No]

Answer

Yes

Question *17.20*

Can we write a function that takes a variable argument list and passes the list to another function (which takes a variable number of arguments)? [Yes/No]

Answer

Yes

Question *17.21*

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

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

void display ( char *s, ... ) ;
void show ( char *t, ... ) ;
int main( )
{
    display ( "Hello", 4, 12, 13, 14, 44 ) ;
    return 0 ;
}
void display ( char *s, ... )
{
    show ( s, ... ) ;
}
void show ( char *t, ... )
{
    va_list ptr ;
    int a ;
    va_start ( ptr, t ) ;
    a = va_arg ( ptr, int ) ;
    printf ( "%f\n", a ) ;
}
```


Answer

The call to *show( )* is improper. This is not the way to pass variable argument list to a function.

Question *17.22*

How will you rewrite program 17.21 such that *show( )* is able to handle variable argument list?

Answer

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

void display ( char *s, ... ) ;
void show ( char *t, va_list ptr1 ) ;
int main( )
{
    display ( "Hello", 4, 12, 13, 14, 44 ) ;
    return 0 ;
}
void display ( char *s, ... )
```

```
{
    va_list ptr ;
    va_start ( ptr, s ) ;
    show ( s, ptr ) ;
}
void show ( char *t, va_list ptr1 )
{
    int a, n, i ;
    a = va_arg ( ptr1, int ) ;
    for ( i = 0 ; i < a ; i++ )
    {
        n = va_arg ( ptr1, int ) ;
        printf ( "%d\n", n ) ;
    }
}
```

Question *17.23*

If I use the following *printf( )* to print a *long int* why I am not warned about the type mismatch?

```
printf ( "%d", num ) ;
```

Answer

When a function accepts a variable number of arguments, its prototype cannot provide any information about the number of arguments and type of those variable arguments. Hence the compiler cannot warn about the mismatches. The programmer must make sure that arguments match or must manually insert explicit typecasts.

Question *17.24*

Can you write a function that works similar to *printf( )*?

Answer

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

void myprintf ( char *, ... ) ;
char *convert ( unsigned int num, int base ) ;
int main( )
{
    int i = 65 ;
    char str[ ] = "Quest Video Courses..IT learning at its best" ;
    myprintf ( "Message = %s %d %x\n", str, i, i ) ;
    return 0 ;
}

void myprintf ( char *fmt, ... )
{
    char *p ;
    int i ;
    unsigned u ;
    char *s ;
    va_list argp ;

    va_start ( argp, fmt ) ;
    p = fmt ;
    for ( p = fmt ; *p != '\0' ; p++ )
    {
        if ( *p != '%' )
        {
            putchar ( *p ) ;
            continue ;
        }
        p++ ;
        switch ( *p )
        {
            case 'c' :
                    i = va_arg ( argp, int ) ;
```

```
                putchar ( i ) ;
                break ;
        case 'd' :
                i = va_arg ( argp, int ) ;
                if ( i < 0 )
                {
                    i = -i ;
                    putchar ( '-' ) ;
                }
                puts ( convert ( i, 10 ) ) ;
                break ;
        case 'o' :
                u = va_arg ( argp, unsigned int ) ;
                puts ( convert ( u, 8 ) ) ;
                break ;
        case 's' :
                s = va_arg ( argp, char * ) ;
                puts ( s ) ;
                break ;
        case 'u' :
                u = va_arg ( argp, unsigned int ) ;
                puts ( convert ( u, 10 ) ) ;
                break ;
        case 'x' :
                u = va_arg ( argp, unsigned int ) ;
                puts ( convert ( u, 16 ) ) ;
                break ;
        case '%' :
                putchar ( '%' ) ;
                break ;
    }
  }
  va_end ( argp ) ;

}

char *convert ( unsigned int num, int base )
{
```



```
  static char buff[33] ;
  char *ptr ;
  ptr = &buff [ sizeof ( buff ) - 1 ] ;
  *ptr = '\0' ;
  do
  {
      *--ptr = "0123456789abcdef"[ num % base ] ;
      num /= base ;
  } while ( num != 0 ) ;
  return ptr ;
}
```

Question *17.25*

How can you concatenate all the variable arguments in a list, which are of character array type?

Answer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

char *vstrcat ( char *first, ... ) ;
int main( )
{
    char *result ;
    result = vstrcat ( "Reena ", "Suyash ", "Mita ", "Amita ", "Namita ",
                       "Piyush ", "Mayur ", "Veena ", "Jay ", "\0" ) ;
    printf ( "%s\n" , result ) ;
    return 0 ;
}

char *vstrcat ( char *first, ... )
{
    int len, noofstrings = 0 ;
```

```
    char *p, *retbuf ;
    va_list argp ;

    if ( first == NULL )
        return NULL ;

    len = strlen ( first ) ;
    va_start ( argp, first ) ;
    p = first ;
    while ( *p != '\0' )
    {
        p = va_arg ( argp , char * ) ;
        len += strlen ( p ) ;
        noofstrings++ ;
    }
    va_end ( argp ) ;

    retbuf = ( char * ) malloc ( len + 1 ) ;
    if ( retbuf == NULL )
        return NULL ;

    strcpy ( retbuf , first ) ;
    va_start ( argp , first ) ;
    while ( noofstrings )
    {
        strcat ( retbuf , p ) ;
        p = va_arg ( argp , char * ) ;
        noofstrings-- ;
    }
    va_end ( argp ) ;
    return retbuf ;
}
```

Question *17.26*

How can a called function determine the number of arguments that have been passed to it?

Answer

It cannot. Any function that takes a variable number of arguments must be able to determine the number of arguments from the arguments themselves. For example, the *printf( )* function does this by looking for format specifiers (*%d* etc.) in the format string. This is the reason why such functions fail badly if there is a mismatch in the format specifiers and the argument list.

If the arguments passed are all of same type we can pass a sentinel value like -1 or 0 or a NULL pointer at the end of the variable argument list. Alternately, we can also pass the count of number of variable arguments.

Question *17.27*

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

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

void display ( char *s, ... ) ;
int fun1( ) ;
int fun2( ) ;

int main( )
{
    int ( *p1 )( ) ;
    int ( *p2 )( ) ;
    p1 = fun1 ;
    p2 = fun2 ;
    display ( "Bye", p1, p2 ) ;
    return 0 ;
}

void display ( char *s, ... )
```

```
{
    int ( *pp1 )( ) ;
    int ( *pp2 )( ) ;
    va_list ptr ;

    va_start ( ptr, s ) ;
    pp1 = va_arg ( ptr, int ( * )( ) ) ;
    ( *pp1 )( ) ;
    pp2 = va_arg ( ptr, int ( * )( ) ) ;
    ( *pp2 )( ) ;
}

int fun1( )
{
    printf ( "Hello\n" ) ;
    return 0 ;
}

int fun2( )
{
    printf ( "Hi\n" ) ;
    return 0 ;
}
```

Answer

va_arg cannot extract the function pointer from the variable argument list in the manner tried here.

Question *17.28*

How will you rectify the error in the program 17.27 above?

Answer

Use *typedef* as shown below:


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

void display ( char *s, ... ) ;
int fun1( ) ;
int fun2( ) ;

int main( )
{
    int ( *p1 )( ) ;
    int ( *p2 )( ) ;
    p1 = fun1 ;
    p2 = fun2 ;
    display ( "Bye", p1, p2 ) ;
    return 0 ;
}

void display ( char *s, ... )
{
    int ( *pp1 )( ), ( *pp2 )( ) ;
    va_list ptr ;
    typedef int ( *funcptr )( ) ;
    va_start ( ptr, s ) ;
    pp1 = va_arg ( ptr, funcptr ) ;
    ( *pp1 )( ) ;
    pp2 = va_arg ( ptr, funcptr ) ;
    ( *pp2 )( ) ;
}

int fun1( )
{
    printf ( "Hello\n" ) ;
    return 0 ;
}

int fun2( )
{
    printf ( "Hi\n" ) ;
```

```
    return 0 ;
}
```

Question *17.29*

What will be the output of the following program?

```
#include <stdio.h>
#include <stdarg.h>
void fun1 ( char, int, int *, float *, char * ) ;
void fun2 ( char ch, ... ) ;
void ( *p1 ) ( char, int, int *, float *, char * ) ;
void ( *p2 ) ( char, ... ) ;
int main( )
{
    char ch = 'A' ; int i = 10 ; float f = 3.14 ; char *p = "Hello" ;
    p1 = fun1 ;
    p2 = fun2 ;
    ( * p1 ) ( ch, i, &i, &f, p ) ;
    ( * p2 ) ( ch, i, &i, &f, p ) ;
    return 0 ;
}

void fun1 ( char ch, int i, int *pi, float *pf, char *p )
{
    printf ( "%c %d %d %f %s\n", ch, i, *pi, *pf, p ) ;
}

void fun2 ( char ch, ... )
{
    int i ; int *pi ; float *pf ; char *p ;
    va_list list ;
    printf ( "%c", ch ) ;
    va_start ( list, ch ) ;
    i = va_arg ( list, int ) ;
    printf ( " %d", i ) ;
    pi = va_arg ( list, int * ) ;
```



```
    printf ( " %d", *pi ) ;
    pf = va_arg ( list, float * ) ;
    printf ( " %f", *pf ) ;
    p = va_arg ( list,char * ) ;
    printf ( " %s\n", p ) ;
}
```

Answer

```
A 10 10 3.140000 Hello
A 10 10 3.140000 Hello
```

Question *17.30*

What will be the output of the following program?

```
#include <stdio.h>
#include <stdarg.h>
void dumplist ( int, ... ) ;
int main( )
{
    dumplist ( 2, 4, 8 );
    dumplist ( 3, 6, 9, 7 );
    return 0 ;
}
void dumplist ( int n, ... )
{
    va_list p ; int i ;
    va_start ( p, n ) ;

    while ( n-- > 0)
    {
        i = va_arg ( p, int ) ;
        printf ( "%d ", i ) ;
    }
    printf ( "\n" ) ;
    va_end ( p ) ;
```

```
}
```

Answer

```
4 8
6 9 7
```

Question *17.31*

State True or False:

Which of the following is correct about the function that receives variable number of arguments?

A. For a function that receives variable number of arguments it is necessary that the function should receive at least one fixed argument.

B. A function that receives variable number of arguments should use *va_arg( )* to extract arguments from the variable argument list.

C. A function that receives variable number of arguments should use *va_end( )* to extract the last argument from the variable argument list.

D. In a function that receives variable number of arguments the fixed arguments passed to the function can be at the end of the argument list.

E. The macro *va_arg* is used to extract an argument from the fixed argument list and advance the pointer to the next argument.

F. It is necessary to call the macro *va_end* if *va_start* is called in the function.

Answer

A. True
B. Tue
C. False
D. False
E. False
F. True