CENG 230

Introduction to C Programming

Week 9 – Functions

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli, and the lecture notes of the textbook by Hanly and Koffman.

Previously on CEng 230!

Homework

- Write a program to read in numbers until the number -1 is encountered. The sum, max and min of all numbers read until this point should be printed out.

Previously on CEng 230!

Modular programming with functions

Modular programming

Previously on CEng 230!

"Experience has shown that the best way to develop and maintain a large program is to construct it from **smaller pieces** or **modules, each** of which is more manageable than the original program.

This technique is called **divide and conquer."**

Function **definition**

Previously on CEng 230!

```
return_type function_name(parameter declarations)
{
  statement-1;
  statement-2;
…
}
```

- if is *return_type* not void, “return” statement has to be used:

```
return expression;
```

Function declaration

Previously on CEng230!

- *return_type* function_name(list-of-params);
- The parameters have to have the same types as in the function definition although the names of the parameters may differ.
- Example:
  - int factorial(int N);
  - void print_matrix(int matrix[N][M]);
- If a function is used before it is defined, it has to be declared first.

Function call

Previously on CEng 230!

function_name(list of arguments)

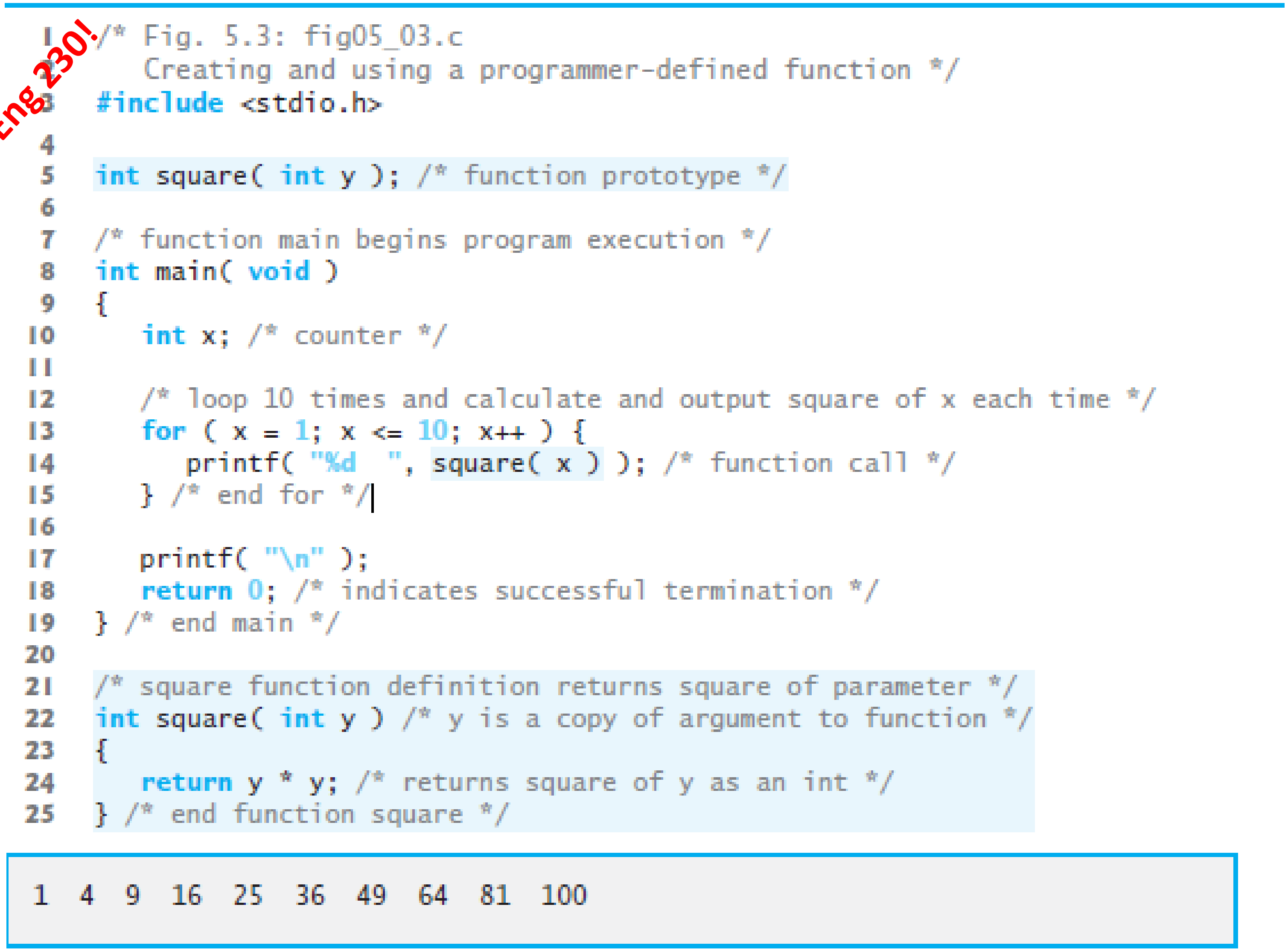
- Example:
  - Function declaration:

    ```
    int greatest(int A, int B, int C);
    ```

  - Example function call:

    ```
    printf(“%d\n”, greatest(10, 20, -10));
    ```

Previously on CEng 230!

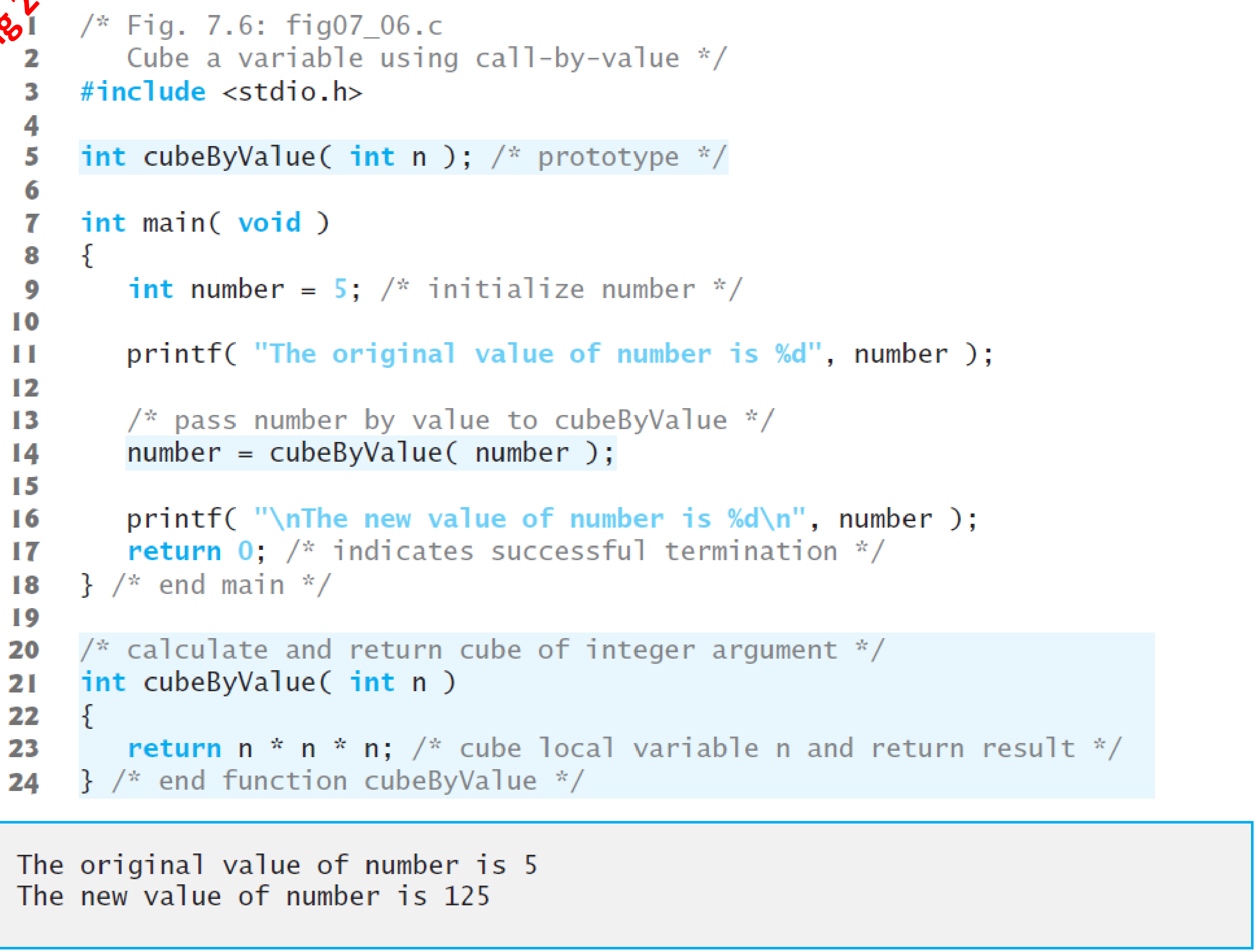


```c
/* Fig. 5.3: fig05_03.c
   Creating and using a programmer-defined function */
#include <stdio.h>

int square( int y ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   int x; /* counter */

   /* loop 10 times and calculate and output square of x each time */
   for ( x = 1; x <= 10; x++ ) {
      printf( "%d  ", square( x ) ); /* function call */
   } /* end for */

   printf( "\n" );
   return 0; /* indicates successful termination */
} /* end main */

/* square function definition returns square of parameter */
int square( int y ) /* y is a copy of argument to function */
{
   return y * y; /* returns square of y as an int */
} /* end function square */
```

```
1  4  9  16  25  36  49  64  81  100
```

Fig. 5.3 | Using a programmer-defined function. (Part 2 of 2.)

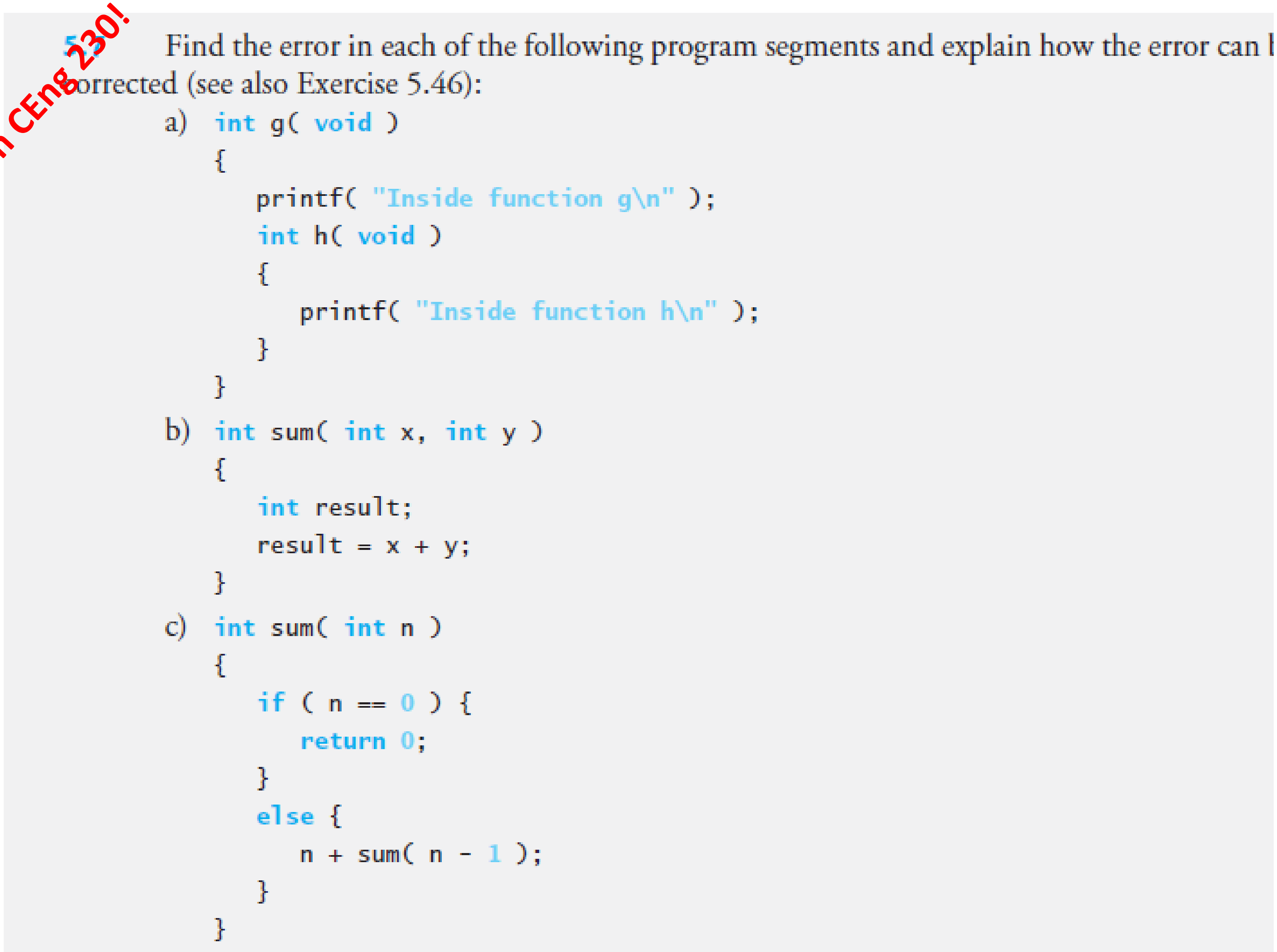
Previously on CEng 230!



```c
/* Fig. 7.6: fig07_06.c
   Cube a variable using call-by-value */
#include <stdio.h>

int cubeByValue( int n ); /* prototype */

int main( void )
{
   int number = 5; /* initialize number */

   printf( "The original value of number is %d", number );

   /* pass number by value to cubeByValue */
   number = cubeByValue( number );

   printf( "\nThe new value of number is %d\n", number );
   return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
   return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

```
The original value of number is 5
The new value of number is 125
```

Previously on CEng 230!



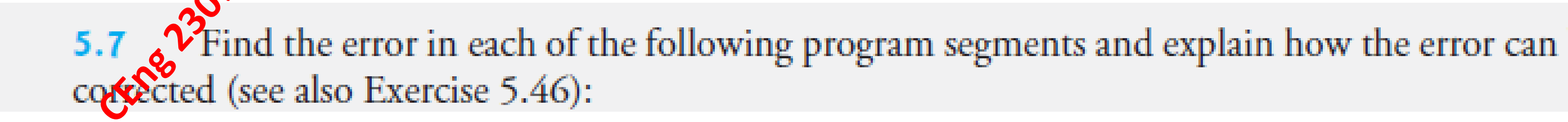
5.7 Find the error in each of the following program segments and explain how the error can corrected (see also Exercise 5.46):

a)
```
int g( void )
{
   printf( "Inside function g\n" );
   int h( void )
   {
      printf( "Inside function h\n" );
   }
}
```

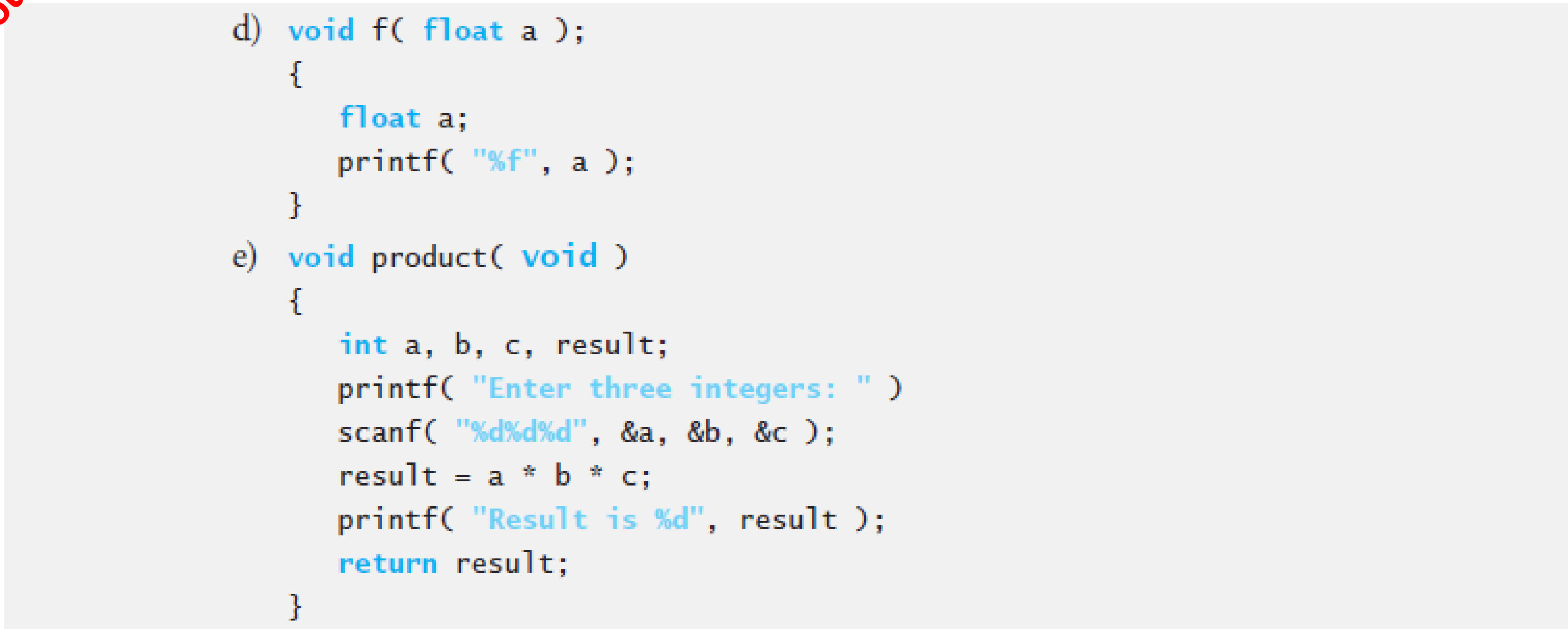
b)
```
int sum( int x, int y )
{
   int result;
   result = x + y;
}
```

c)
```
int sum( int n )
{
   if ( n == 0 ) {
      return 0;
   }
   else {
      n + sum( n - 1 );
   }
}
```

Previously on CEng 230!



5.7 Find the error in each of the following program segments and explain how the error can corrected (see also Exercise 5.46):

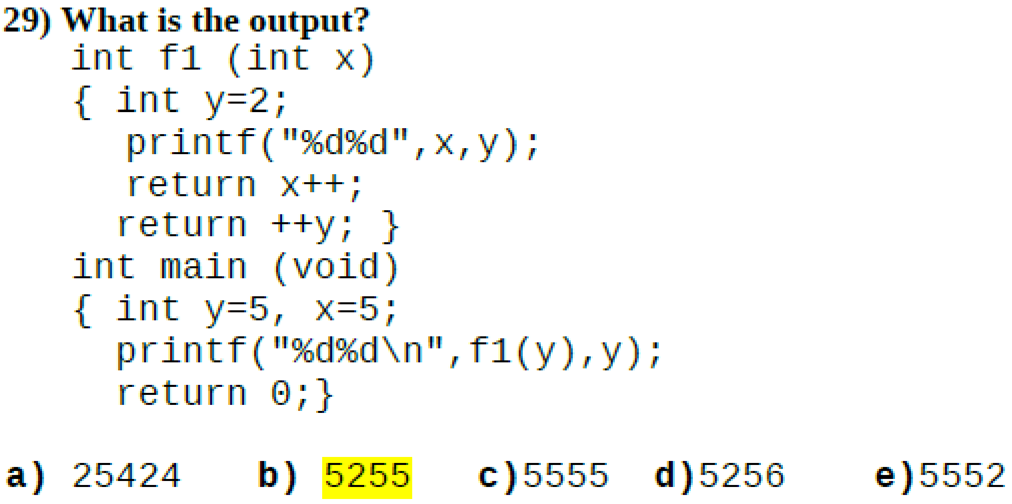


d)
```
void f( float a );
{
   float a;
   printf( "%f", a );
}
```

e)
```
void product( void )
{
   int a, b, c, result;
   printf( "Enter three integers: " )
   scanf( "%d%d%d", &a, &b, &c );
   result = a * b * c;
   printf( "Result is %d", result );
   return result;
}
```

Previously on CEng 230!

Sample 1



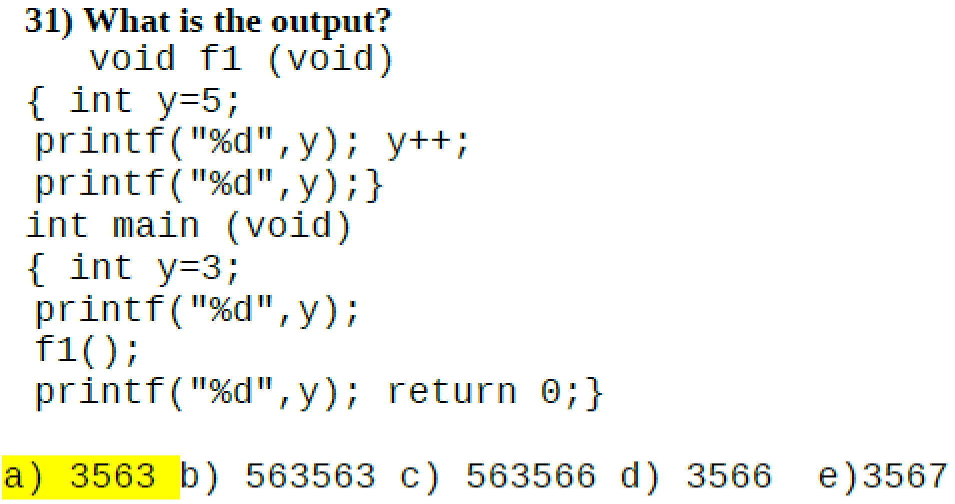
29) What is the output?

```
int f1 (int x)
{ int y=2;
   printf("%d%d",x,y);
   return x++;
  return ++y; }
int main (void)
{ int y=5, x=5;
  printf("%d%d\n",f1(y),y);
  return 0;}
```

a) 25424 **b)** 5255 **c)**5555 **d)**5256 **e)**5552

Previously on CEng 230!

Sample 2

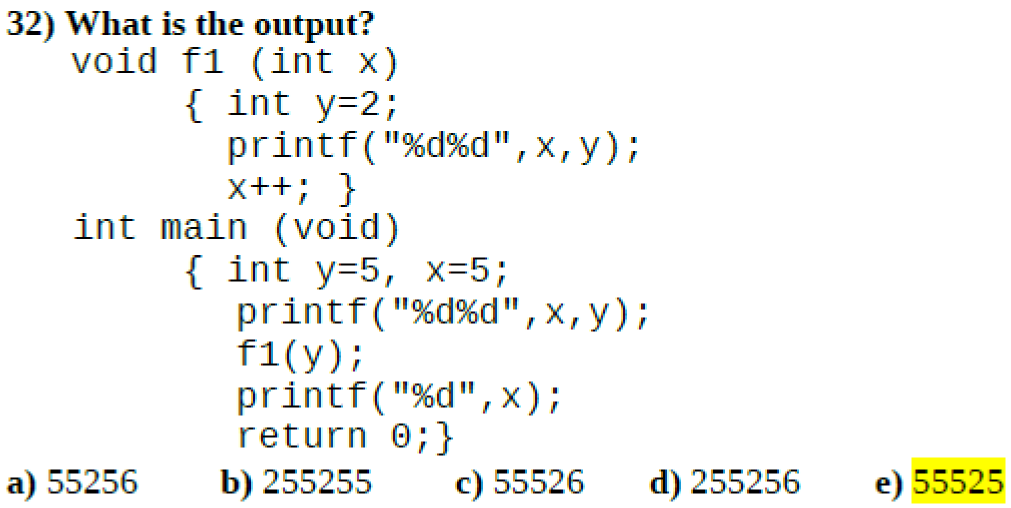


31) What is the output?

```
   void f1 (void)
{ int y=5;
 printf("%d",y); y++;
 printf("%d",y);}
int main (void)
{ int y=3;
 printf("%d",y);
 f1();
 printf("%d",y); return 0;}
```

a) 3563 b) 563563 c) 563566 d) 3566 e)3567

Sample 3



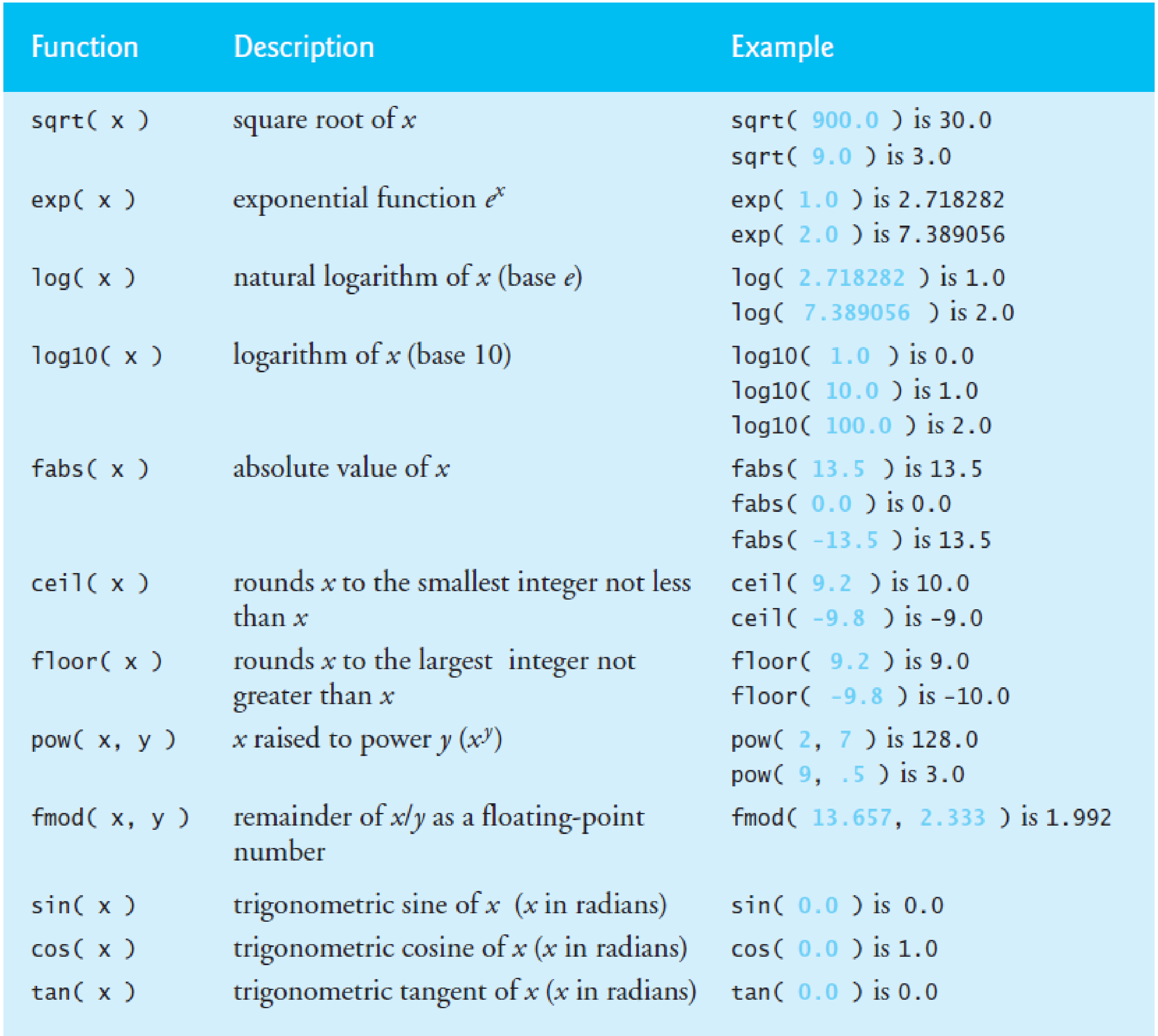
32) What is the output?

```
void f1 (int x)
     { int y=2;
       printf("%d%d",x,y);
       x++; }
int main (void)
     { int y=5, x=5;
       printf("%d%d",x,y);
       f1(y);
       printf("%d",x);
       return 0;}
```

a) 55256 **b)** 255255 **c)** 55526 **d)** 255256 **e)** 55525

Today

- Built-in functions
  - Math library (#include<math.h>)
  - Stdlib library (#include<stdlib.h>)



Function	Description	Example

`sqrt( x )`	square root of *x*	`sqrt( 900.0 )` is `30.0`
		`sqrt( 9.0 )` is `3.0`

`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.718282`
		`exp( 2.0 )` is `7.389056`

`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0`
		`log( 7.389056 )` is `2.0`

`log10( x )`	logarithm of *x* (base 10)	`log10( 1.0 )` is `0.0`
		`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

`fabs( x )`	absolute value of *x*	`fabs( 13.5 )` is `13.5`
		`fabs( 0.0 )` is `0.0`
		`fabs( -13.5 )` is `13.5`

`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`pow( x, y )`	*x* raised to power *y* (x^y)	`pow( 2, 7 )` is `128.0`
		`pow( 9, .5 )` is `3.0`

`fmod( x, y )`	remainder of *x*/*y* as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0.0`
`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0.0`

Fig. 5.2 | Commonly used math library functions.

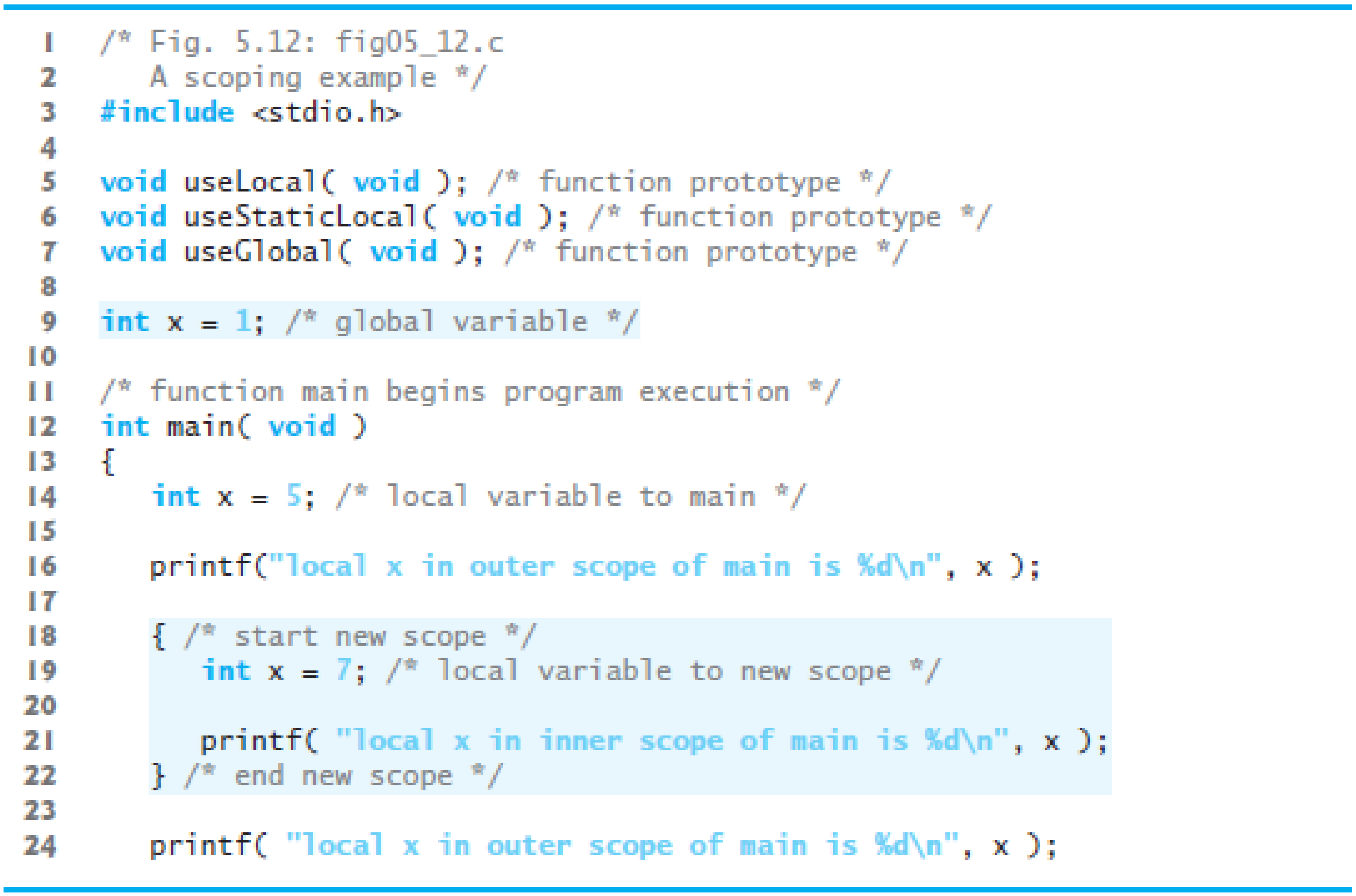
#include<math.h>

```c
/* Fig. 5.7: fig05_07.c
   Shifted, scaled integers produced by 1 + rand() % 6 */
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
   int i; /* counter */

   /* loop 20 times */
   for ( i = 1; i <= 20; i++ ) {

      /* pick random number from 1 to 6 and output it */
      printf( "%10d", 1 + ( rand() % 6 ) );

      /* if counter is divisible by 5, begin new line of output */
      if ( i % 5 == 0 ) {
         printf( "\n" );
      } /* end if */
   } /* end for */

   return 0; /* indicates successful termination */
} /* end main */
```

```
         6         6         5         5         6
         5         1         1         5         3
         6         6         2         4         2
         6         2         3         4         1
```

Fig. 5.7 | Shifted, scaled random integers produced by 1 + rand() % 6. (Part 2 of 2.)

Scope



```c
/* Fig. 5.12: fig05_12.c
   A scoping example */
#include <stdio.h>

void useLocal( void ); /* function prototype */
void useStaticLocal( void ); /* function prototype */
void useGlobal( void ); /* function prototype */

int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
   int x = 5; /* local variable to main */

   printf("local x in outer scope of main is %d\n", x );

   { /* start new scope */
      int x = 7; /* local variable to new scope */

      printf( "local x in inner scope of main is %d\n", x );
   } /* end new scope */

   printf( "local x in outer scope of main is %d\n", x );
```

Fig. 5.12 | Scoping example. (Part 1 of 3.)

Scope Rules

- File scope
  - Identifier defined outside function, known in all functions
  - Used for global variables, function definitions, function prototypes
- Function scope
  - Can only be referenced inside a function body

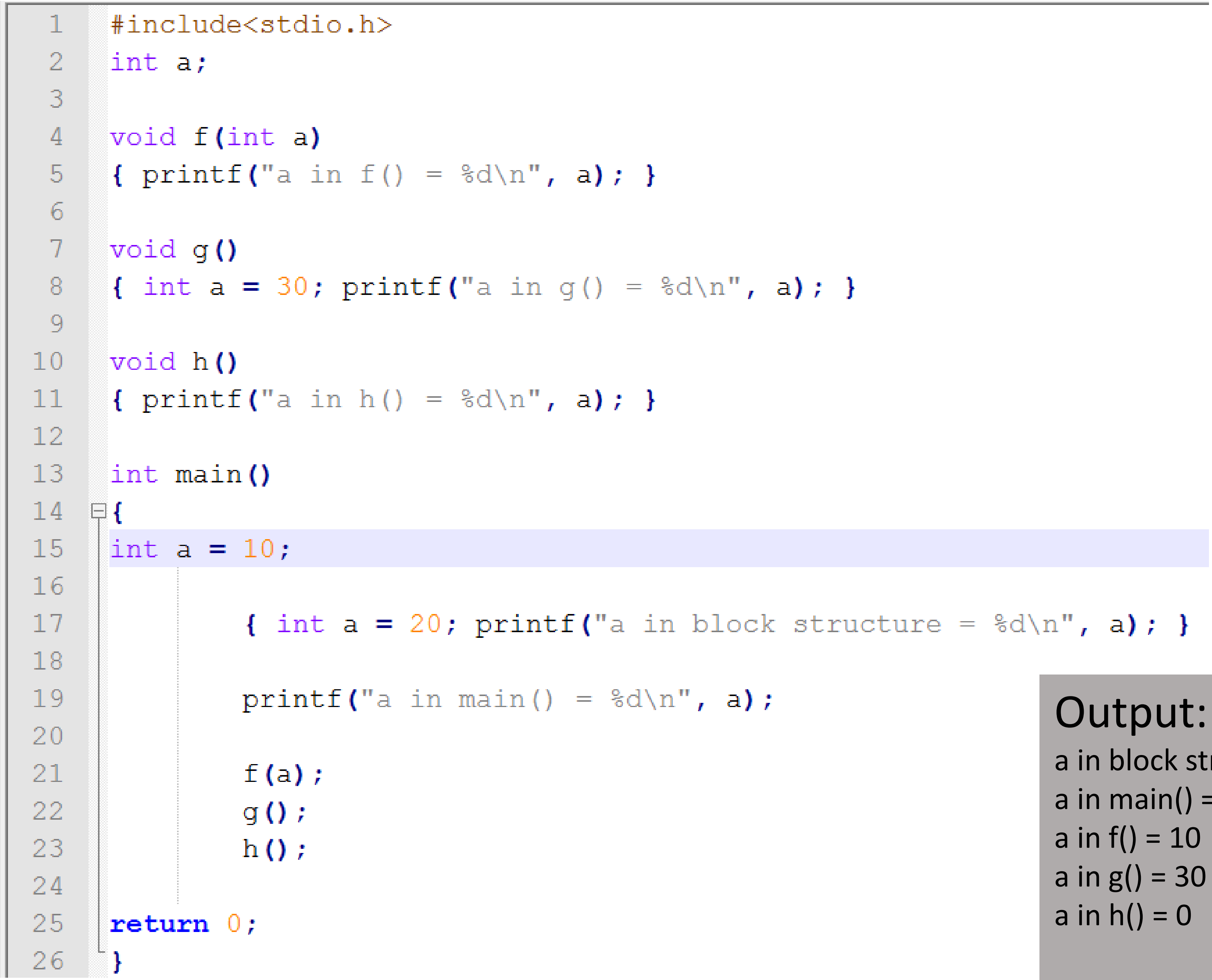
Scope Rules

- Block scope
    - Identifier declared inside a block
        - Block scope begins at definition, ends at right brace
    - Used for variables, function parameters (local variables of function)
    - Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block
- Function prototype scope
    - Used for identifiers in parameter list

Namespaces

- Determines where the definition of variables are valid!
- Global space.
- main() function space.
- Block structures.

Namespace Example



```c
#include<stdio.h>
int a;

void f(int a)
{ printf("a in f() = %d\n", a); }

void g()
{ int a = 30; printf("a in g() = %d\n", a); }

void h()
{ printf("a in h() = %d\n", a); }

int main()
{
int a = 10;

        { int a = 20; printf("a in block structure = %d\n", a); }

        printf("a in main() = %d\n", a);

        f(a);
        g();
        h();

return 0;
}
```

Output:

```
a in block structure = 20
a in main() = 10
a in f() = 10
a in g() = 30
a in h() = 0
```

Storage-based Types of Variables

Auto vs. register vs. static variables

Storage Classes

- Storage class specifiers
  - Storage duration – how long an object exists in memory
  - Scope – where object can be referenced in program
  - Linkage – specifies the files in which an identifier is known (more in Chapter 14)
- Automatic storage
  - Object created and destroyed within its block
  - `auto`: default for local variables

    ```
    auto double x, y;
    ```
  - `register`: tries to put variable into high-speed registers
    - Can only be used for automatic variables

      ```
      register int counter = 1;
      ```

Storage Classes

- Static storage
  - Variables exist for entire program execution
  - Default value of zero
  - `static`: local variables defined in functions.
    - Keep value after function ends
    - Only known in their own function
  - `extern`: default for global variables and functions
    - Known in any function

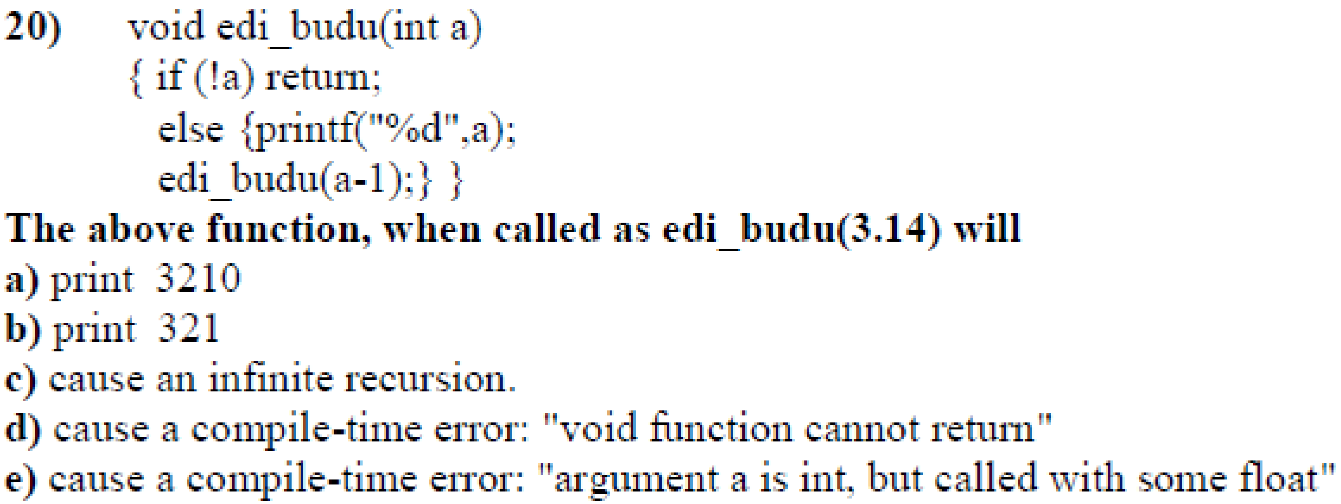
Parameter passing in functions

Call by Value

- The arguments of the function are just copies of the passed data!

```
void f(int a)
{
   a = 10 * a;
}
void g(int b)
{
   b = 10;
   f(b);
   printf(“%d”,  b);
}
```

Sample 4



20)
```
void edi_budu(int a)
{ if (!a) return;
  else {printf("%d",a);
  edi_budu(a-1);} }
```

The above function, when called as edi_budu(3.14) will

a) print 3210

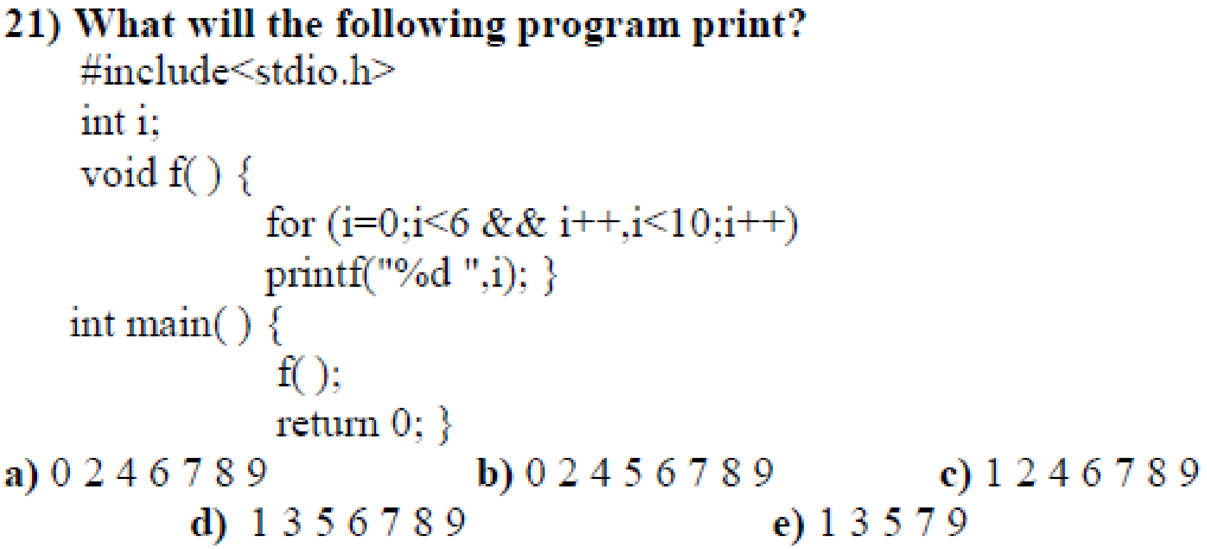
b) print 321

c) cause an infinite recursion.

d) cause a compile-time error: "void function cannot return"

e) cause a compile-time error: "argument a is int, but called with some float"

Sample 5

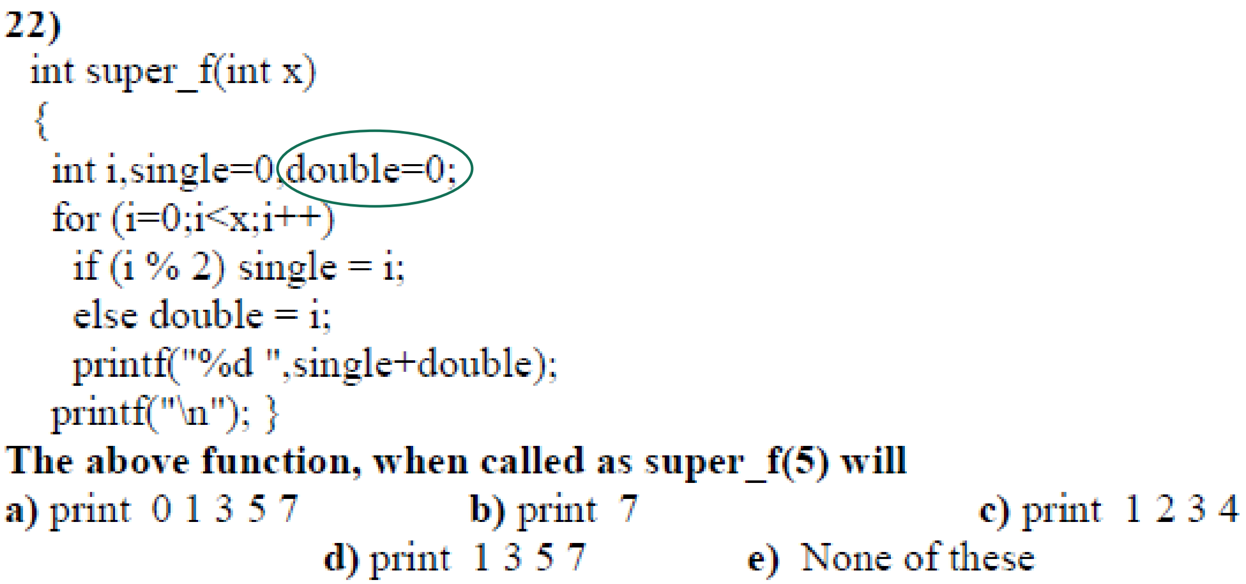


21) What will the following program print?
```
 #include<stdio.h>
 int i;
 void f( ) {
           for (i=0;i<6 && i++,i<10;i++)
           printf("%d ",i); }
int main( ) {
            f( );
            return 0; }
```

a) 0 2 4 6 7 8 9 **b)** 0 2 4 5 6 7 8 9 **c)** 1 2 4 6 7 8 9

d) 1 3 5 6 7 8 9 **e)** 1 3 5 7 9

Sample 6



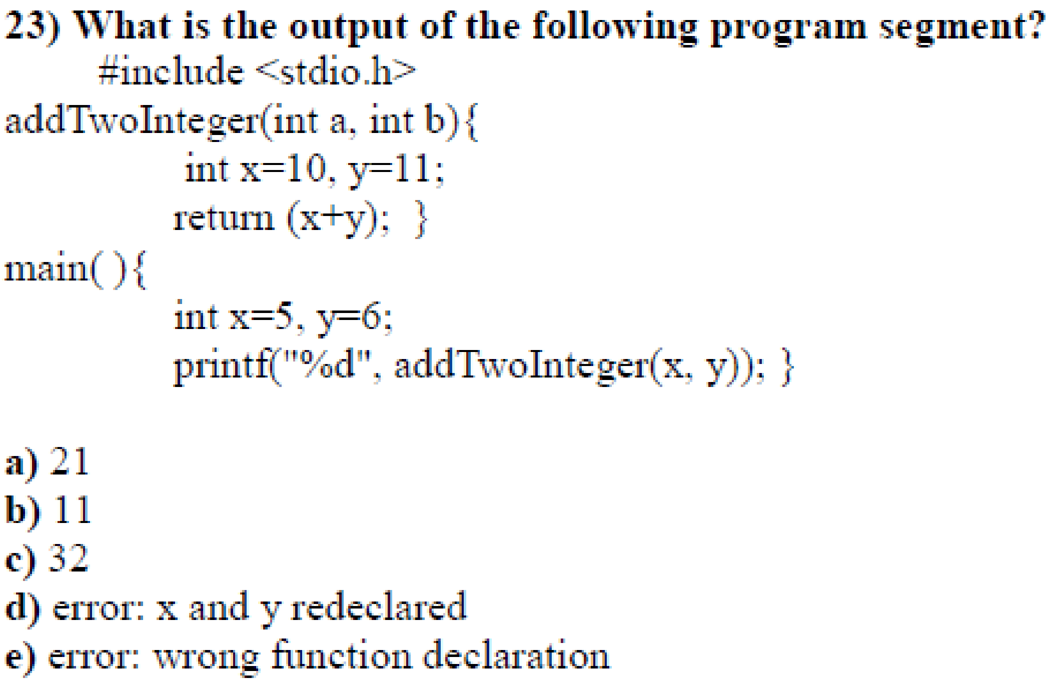
22)

```
 int super_f(int x)
 {
   int i,single=0,double=0;
   for (i=0;i<x;i++)
     if (i % 2) single = i;
     else double = i;
     printf("%d ",single+double);
   printf("\n"); }
```

The above function, when called as super_f(5) will

a) print 0 1 3 5 7 **b)** print 7 **c)** print 1 2 3 4 **d)** print 1 3 5 7 **e)** None of these

Sample 7



23) What is the output of the following program segment?

```
    #include <stdio.h>
addTwoInteger(int a, int b){
          int x=10, y=11;
          return (x+y);  }
main( ){
          int x=5, y=6;
          printf("%d", addTwoInteger(x, y)); }
```

a) 21

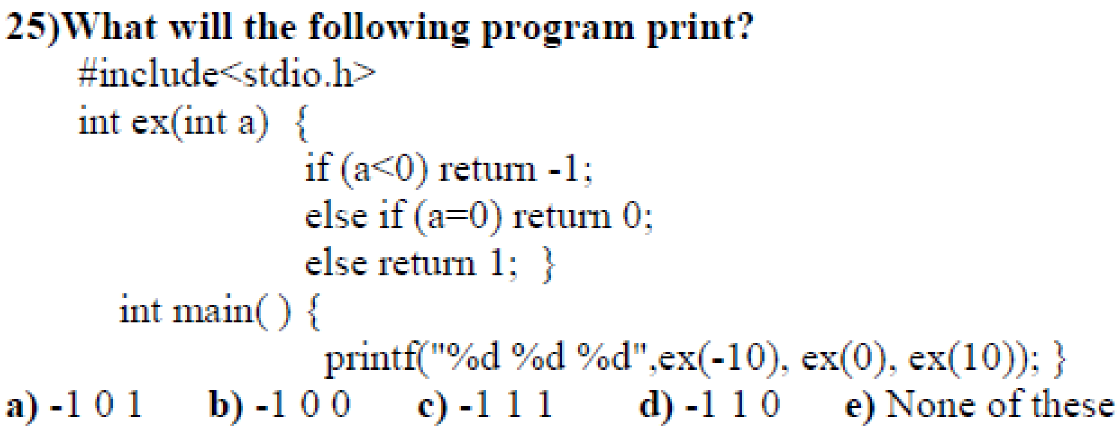
b) 11

c) 32

d) error: x and y redeclared

e) error: wrong function declaration

Sample 8

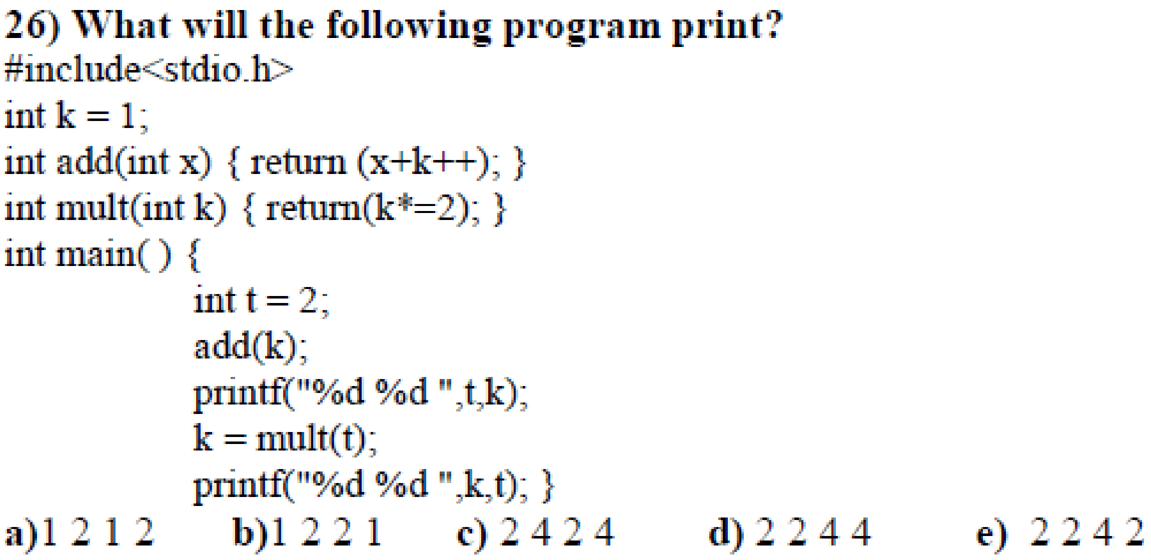


25)What will the following program print?

```
#include<stdio.h>
int ex(int a)  {
                    if (a<0) return -1;
                    else if (a=0) return 0;
                    else return 1;  }
    int main( ) {
                    printf("%d %d %d",ex(-10), ex(0), ex(10)); }
```

a) -1 0 1   **b)** -1 0 0   **c)** -1 1 1   **d)** -1 1 0   **e)** None of these

Sample 9

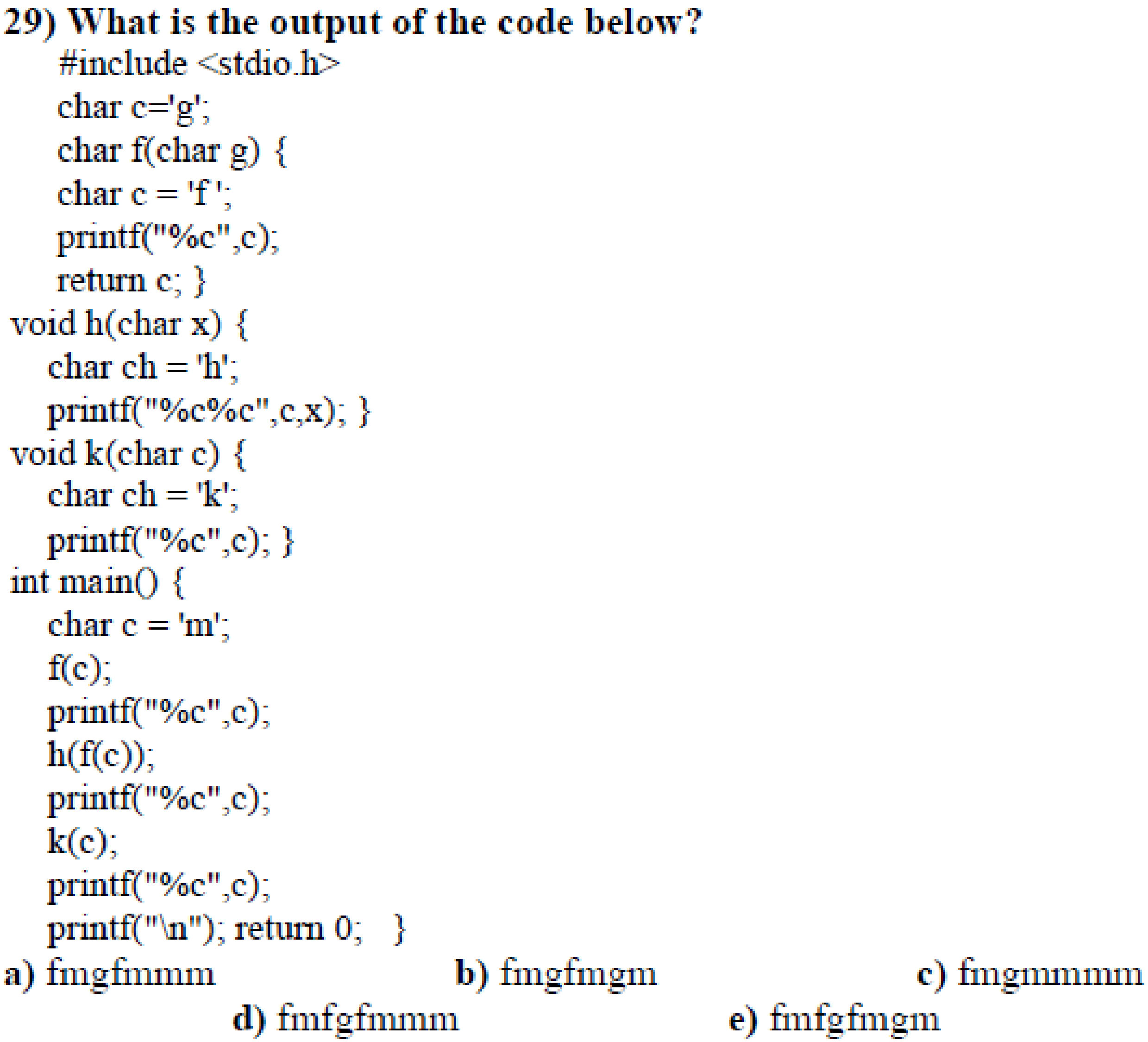


26) What will the following program print?

```
#include<stdio.h>
int k = 1;
int add(int x) { return (x+k++); }
int mult(int k) { return(k*=2); }
int main( ) {
                int t = 2;
                add(k);
                printf("%d %d ",t,k);
                k = mult(t);
                printf("%d %d ",k,t); }
```

a)1 2 1 2   **b)**1 2 2 1   **c)** 2 4 2 4   **d)** 2 2 4 4   **e)** 2 2 4 2

Sample 11



29) What is the output of the code below?

```
     #include <stdio.h>
     char c='g';
     char f(char g) {
     char c = 'f ';
     printf("%c",c);
     return c; }
void h(char x) {
    char ch = 'h';
    printf("%c%c",c,x); }
void k(char c) {
    char ch = 'k';
    printf("%c",c); }
int main() {
    char c = 'm';
    f(c);
    printf("%c",c);
    h(f(c));
    printf("%c",c);
    k(c);
    printf("%c",c);
    printf("\n"); return 0;   }
```

a) fmgfmmm　**b)** fmgfmgm　**c)** fmgmmmm

d) fmfgfmmm　**e)** fmfgfmgm