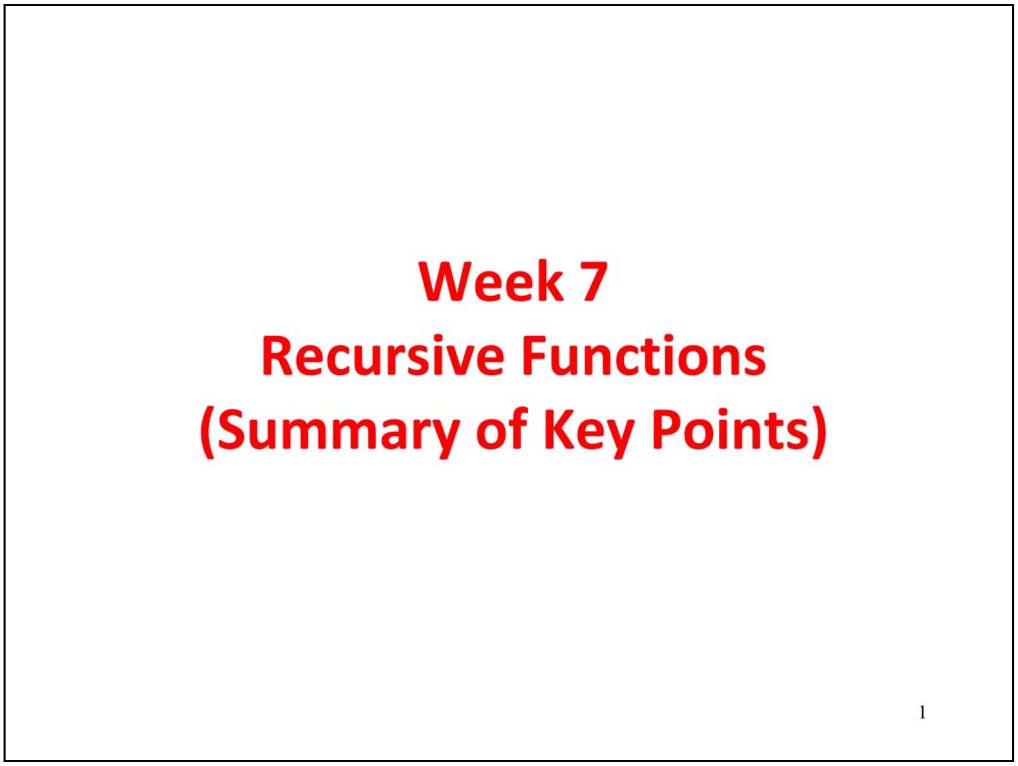
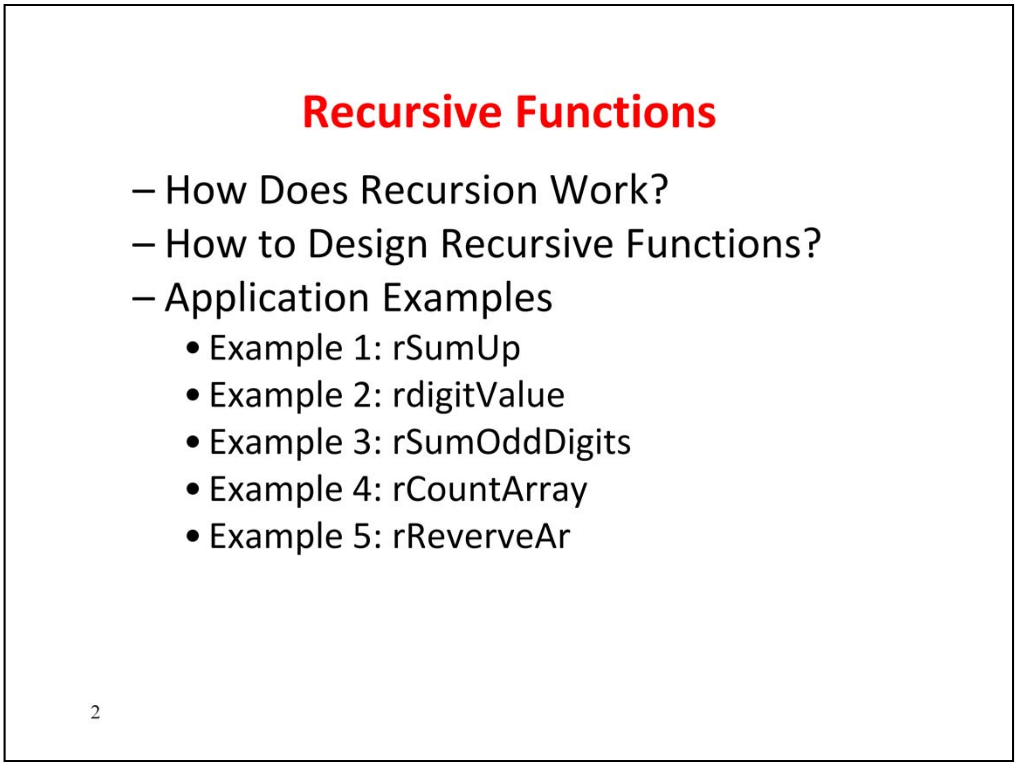
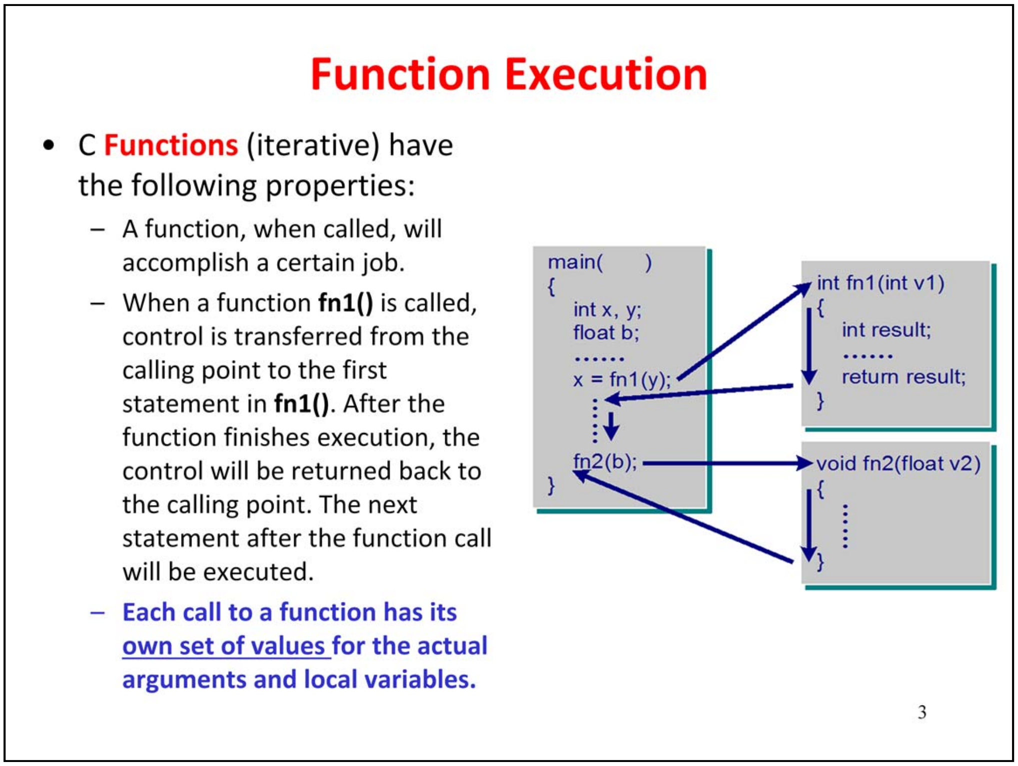


Week 7
Recursive Functions
(Summary of Key Points)



Recursive Functions

- How Does Recursion Work?
- How to Design Recursive Functions?
- Application Examples
  - Example 1: rSumUp
  - Example 2: rdigitValue
  - Example 3: rSumOddDigits
  - Example 4: rCountArray
  - Example 5: rReverveAr



Function Execution

- C **Functions** (iterative) have the following properties:
  - A function, when called, will accomplish a certain job.
  - When a function **fn1()** is called, control is transferred from the calling point to the first statement in **fn1()**. After the function finishes execution, the control will be returned back to the calling point. The next statement after the function call will be executed.
  - **Each call to a function has its own set of values for the actual arguments and local variables.**

```
main(      )
{
   int x, y;
   float b;
   ......
   x = fn1(y);

   fn2(b);
}
```

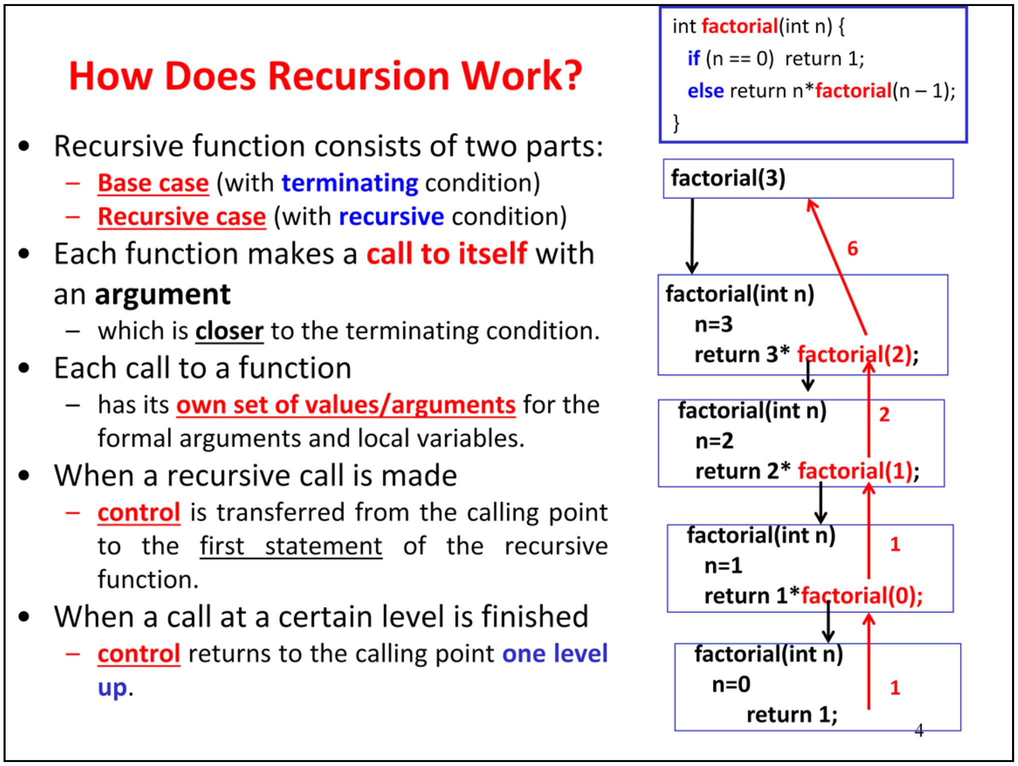
```
int fn1(int v1)
{
   int result;
   ......
   return result;
}
```

```
void fn2(float v2)
{
   ......
}
```

3

What is Recursion?

1. In C, functions have the following properties:
   1) A function, when called, will accomplish a certain job.
   2) When a function **F()** is called, control is transferred from the calling point to the first statement in **F()**. After the function finishes execution, the control will return back to the calling point. The next statement after the function call will be executed.
   3) Each call to a function has its own set of values for the actual arguments and local variables.
2. These properties are important for the understanding of *recursive functions*.



How Does Recursion Work?

- Recursive function consists of two parts:
  - Base case (with terminating condition)
  - Recursive case (with recursive condition)
- Each function makes a call to itself with an argument
  - which is closer to the terminating condition.
- Each call to a function
  - has its own set of values/arguments for the formal arguments and local variables.
- When a recursive call is made
  - control is transferred from the calling point to the first statement of the recursive function.
- When a call at a certain level is finished
  - control returns to the calling point one level up.

```
int factorial(int n) {
  if (n == 0)  return 1;
  else return n*factorial(n – 1);
}
```

factorial(3)

factorial(int n)
n=3
return 3* factorial(2);

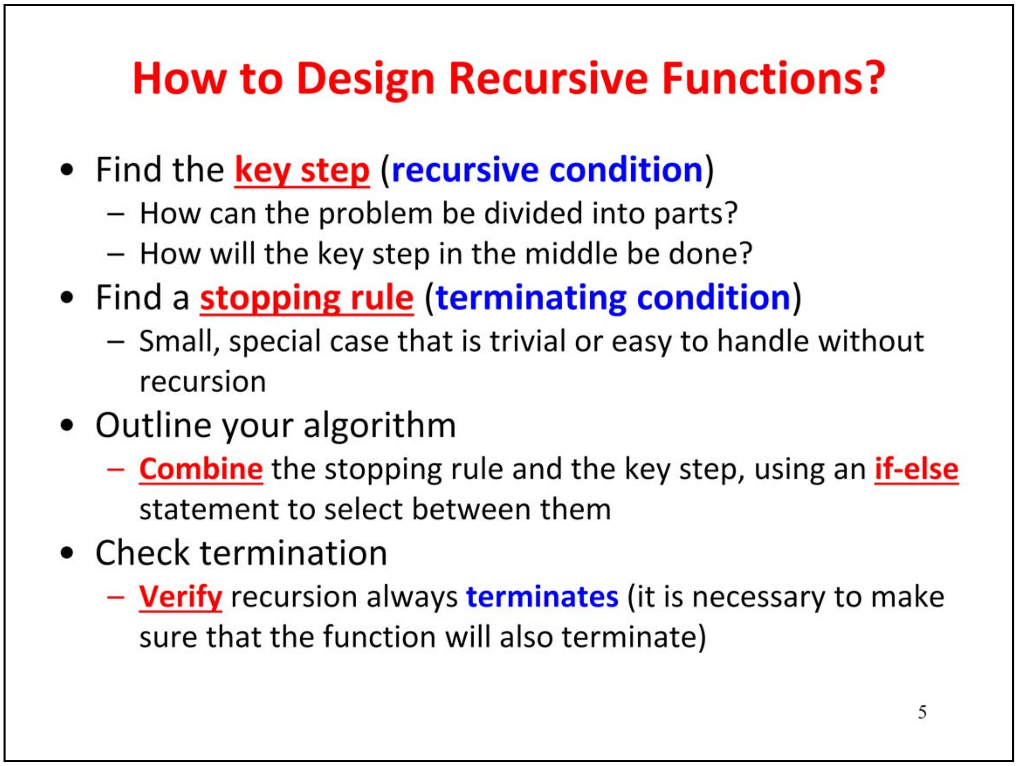
factorial(int n)
n=2
return 2* factorial(1);

factorial(int n)
n=1
return 1*factorial(0);

factorial(int n)
n=0
return 1;

How Does Recursion Work?

1. A recursive function is a function that calls itself.
2. When a function calls itself, the execution of the current function is suspended, the information that it needs to continue execution is saved, and the recursive call is then evaluated.
3. Each function makes a call to itself with an argument which is closer to the terminating condition.
4. Each call to a function has its own set of values/arguments for the formal arguments and local variables, which is independent from the variables that are created from the previous calls to the function.
5. When a recursive call is made, control is transferred from the calling point to the first statement of the recursive function.
6. When a call at a certain level is finished, the control is returned to the calling point one level up. The evaluation result is passed back to the previous call, until the process is completed.



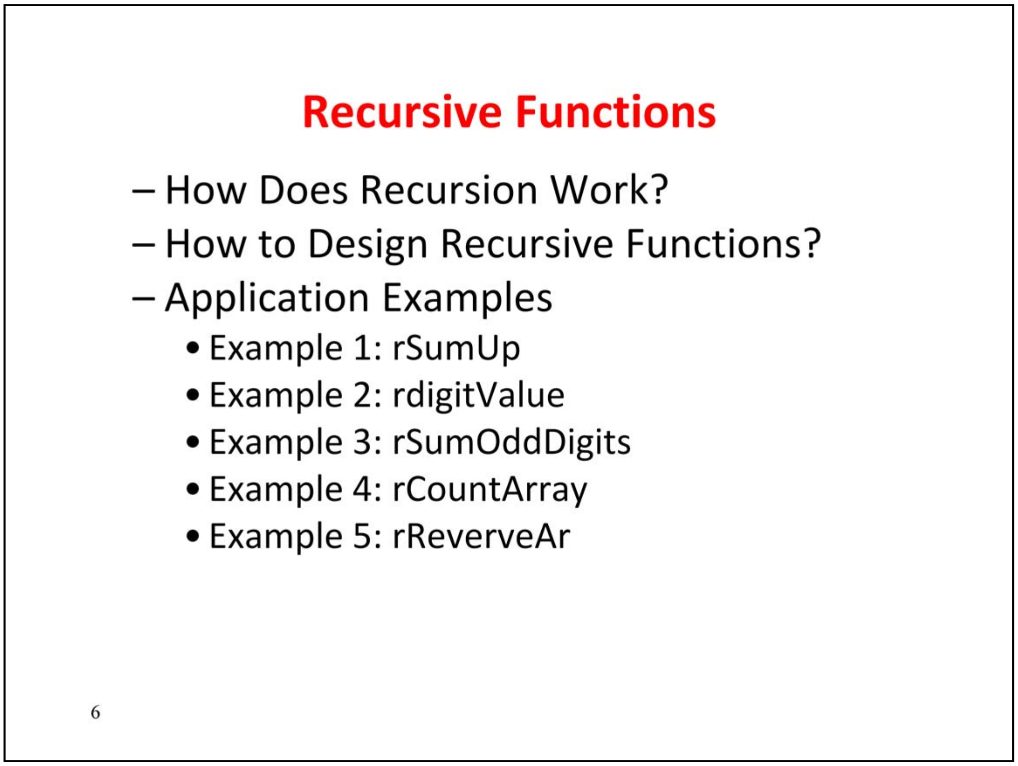
How to Design Recursive Functions?

- Find the **key step** (**recursive condition**)
  - How can the problem be divided into parts?
  - How will the key step in the middle be done?
- Find a **stopping rule** (**terminating condition**)
  - Small, special case that is trivial or easy to handle without recursion
- Outline your algorithm
  - **Combine** the stopping rule and the key step, using an **if-else** statement to select between them
- Check termination
  - **Verify** recursion always **terminates** (it is necessary to make sure that the function will also terminate)

5

How to Design Recursive Functions?

1. There are 4 steps involved when you design any recursive functions:
   1) Find the key step (recursive condition) – To decide how can the problem be divided into parts, and how will the key step in the middle be done.
   2) Find a stopping rule (terminating condition) – To decide the terminating condition which should be small and special case that is trivial or easy to handle without recursion.
   3) Outline your algorithm – To combine the stopping rule and the key step, and use an **if-else** statement to select between them.
   4) Check termination – To verify the recursion always terminates. It is necessary to make sure that the function will also terminate.



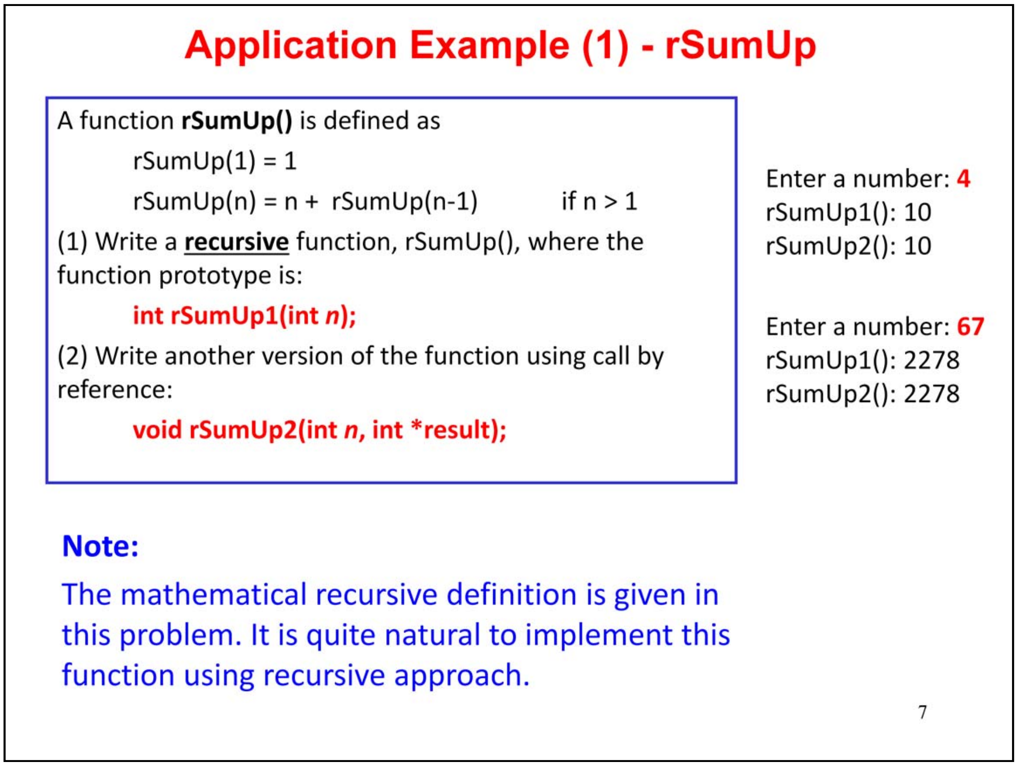
Recursive Functions

- How Does Recursion Work?
- How to Design Recursive Functions?
- Application Examples
  - Example 1: rSumUp
  - Example 2: rdigitValue
  - Example 3: rSumOddDigits
  - Example 4: rCountArray
  - Example 5: rReverveAr

6

Application Examples

1) The applications to be discussed include:
    1) rSumUp
    2) rdigitValue
    3) rSumOddDigits
    4) rCountArray
    5) rReverseAr



Application Example (1) - rSumUp

A function **rSumUp()** is defined as

rSumUp(1) = 1

rSumUp(n) = n + rSumUp(n-1)   if n > 1

(1) Write a **recursive** function, rSumUp(), where the function prototype is:

int rSumUp1(int *n*);

(2) Write another version of the function using call by reference:

void rSumUp2(int *n*, int *result);

```
Enter a number: 4
rSumUp1(): 10
rSumUp2(): 10

Enter a number: 67
rSumUp1(): 2278
rSumUp2(): 2278
```

Note:

The mathematical recursive definition is given in this problem. It is quite natural to implement this function using recursive approach.

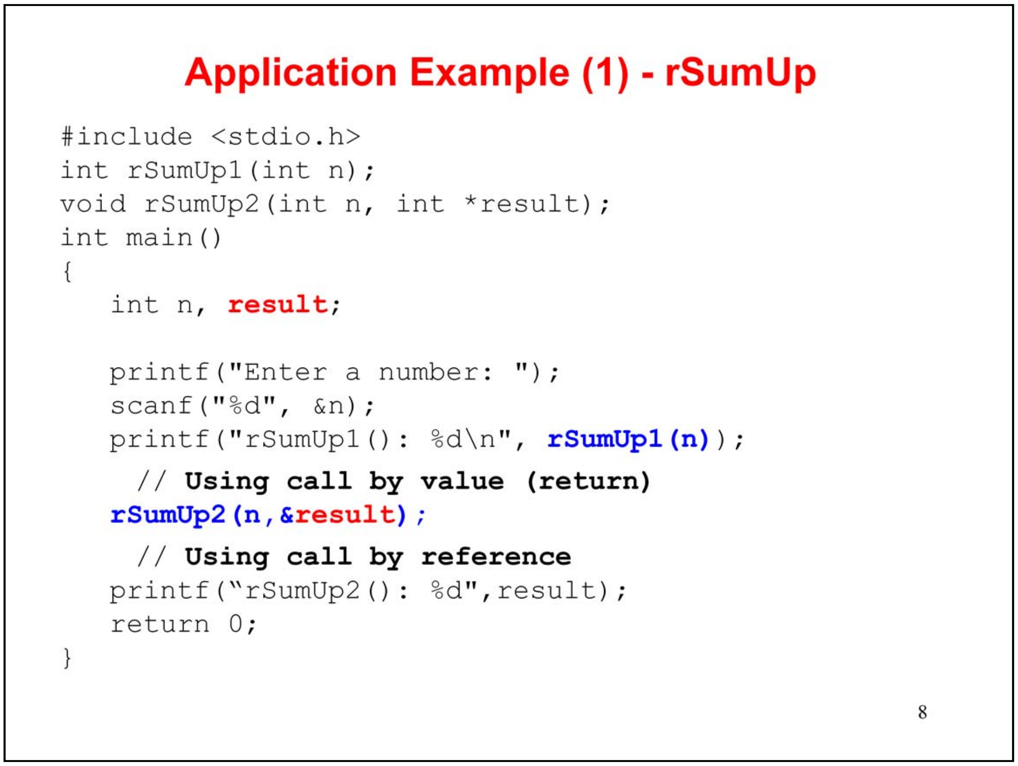
7

Application Example (1) – rSumUp

1. The function prototypes are given below:

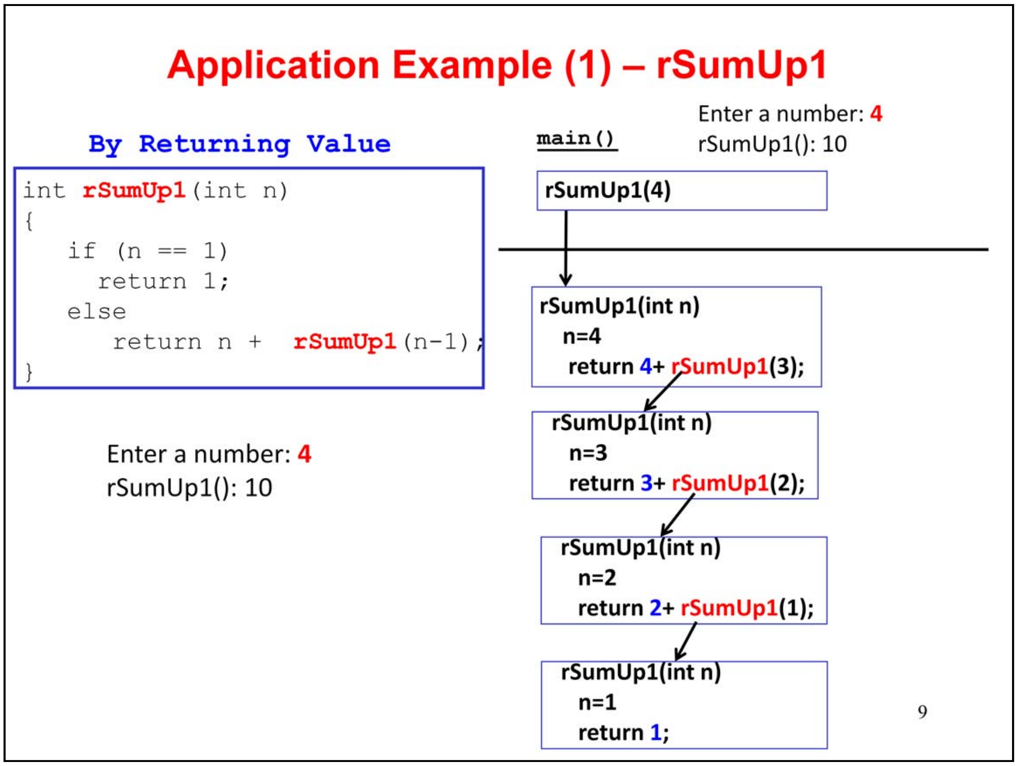
    int **rSumUp1(**int n);

    void **rSumUp2(**int n);



Application Example (1) - rSumUp

```
#include <stdio.h>
int rSumUp1(int n);
void rSumUp2(int n, int *result);
int main()
{
   int n, result;

   printf("Enter a number: ");
   scanf("%d", &n);
   printf("rSumUp1(): %d\n", rSumUp1(n));
     // Using call by value (return)
   rSumUp2(n,&result);
     // Using call by reference
   printf(“rSumUp2(): %d",result);
   return 0;
}
```



Application Example (1) – rSumUp1

By Returning Value

```
int rSumUp1(int n)
{
   if (n == 1)
     return 1;
   else
      return n +  rSumUp1(n-1);
}
```

```
Enter a number: 4
rSumUp1(): 10
```

main()

```
Enter a number: 4
rSumUp1(): 10
```

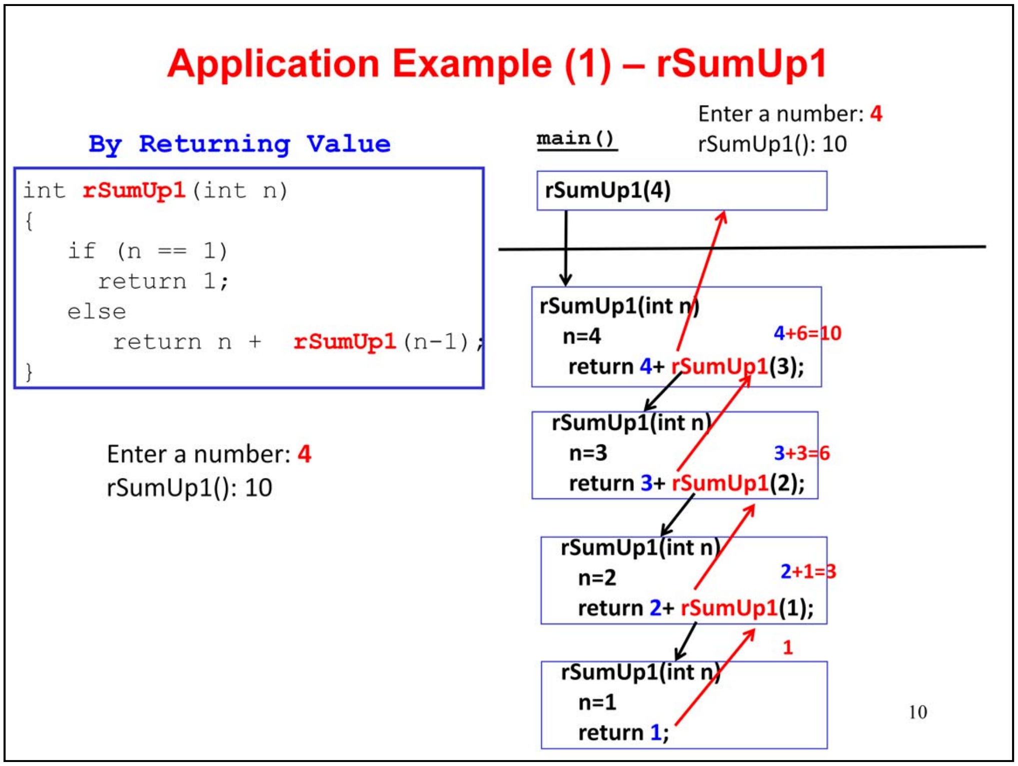
rSumUp1(4)

rSumUp1(int n)
n=4
return 4+ rSumUp1(3);

rSumUp1(int n)
n=3
return 3+ rSumUp1(2);

rSumUp1(int n)
n=2
return 2+ rSumUp1(1);

rSumUp1(int n)
n=1
return 1;

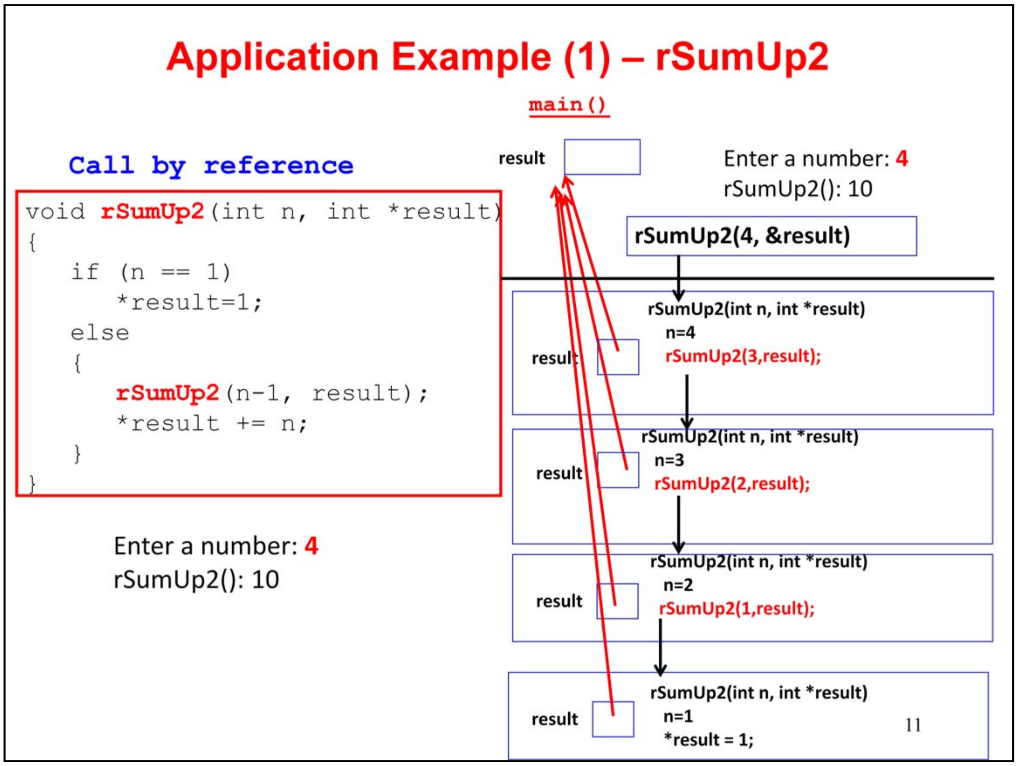


Application Example (1) – rSumUp1

By Returning Value

```
int rSumUp1(int n)
{
   if (n == 1)
     return 1;
   else
      return n +  rSumUp1(n-1);
}
```

Enter a number: 4
rSumUp1(): 10



Application Example (1) – rSumUp2

Call by reference

```
void rSumUp2(int n, int *result)
{
   if (n == 1)
      *result=1;
   else
   {
      rSumUp2(n-1, result);
      *result += n;
   }
}
```

Enter a number: **4**
rSumUp2(): 10

main()

result

Enter a number: **4**
rSumUp2(): 10

rSumUp2(4, &result)

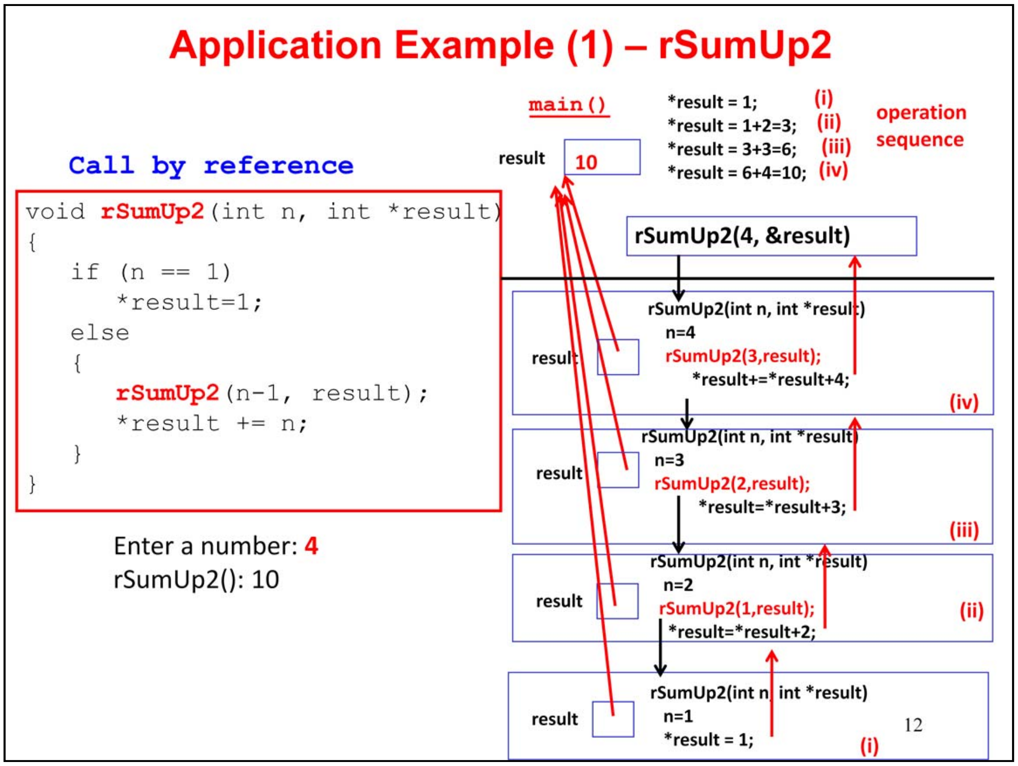
rSumUp2(int n, int *result)
n=4
rSumUp2(3,result);
result

rSumUp2(int n, int *result)
n=3
rSumUp2(2,result);
result

rSumUp2(int n, int *result)
n=2
rSumUp2(1,result);
result

rSumUp2(int n, int *result)
n=1
*result = 1;
result

11



Application Example (1) – rSumUp2

Call by reference

```
void rSumUp2(int n, int *result)
{
   if (n == 1)
      *result=1;
   else
   {
      rSumUp2(n-1, result);
      *result += n;
   }
}
```

```
Enter a number: 4
rSumUp2(): 10
```

main()

result 10

*result = 1; (i)
*result = 1+2=3; (ii)
*result = 3+3=6; (iii)
*result = 6+4=10; (iv)

operation sequence

rSumUp2(4, &result)

```
rSumUp2(int n, int *result)
  n=4
  rSumUp2(3,result);
     *result+=*result+4;
```

(iv)

```
rSumUp2(int n, int *result)
  n=3
  rSumUp2(2,result);
     *result=*result+3;
```

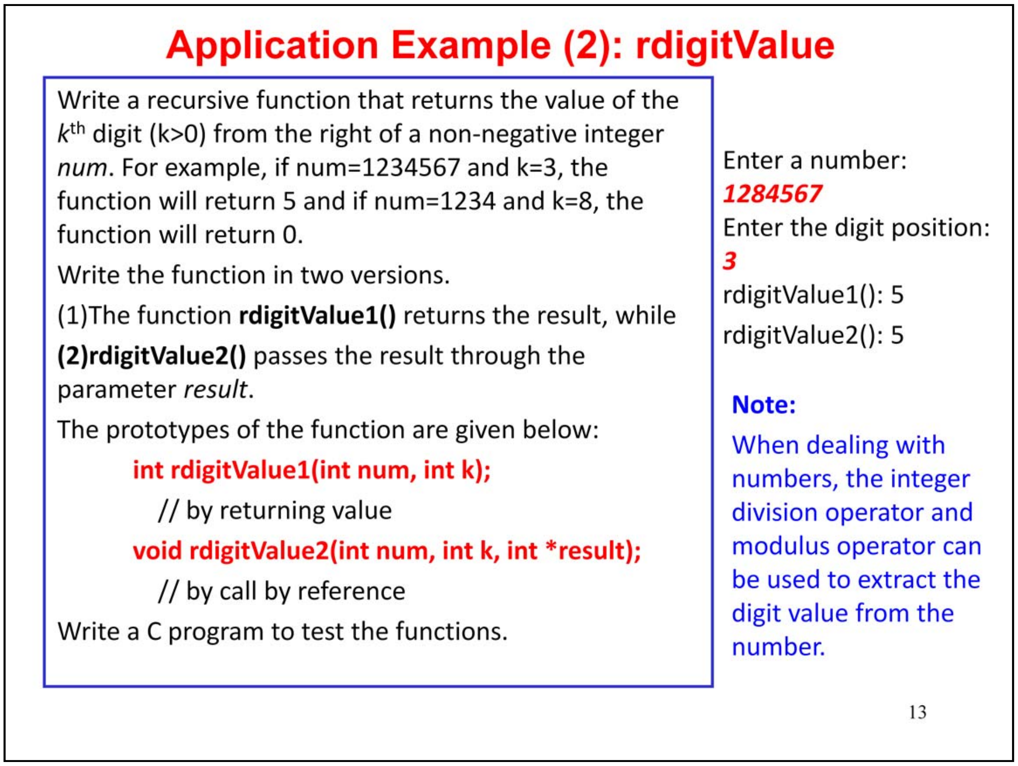
(iii)

```
rSumUp2(int n, int *result)
  n=2
  rSumUp2(1,result);
  *result=*result+2;
```

(ii)

```
rSumUp2(int n, int *result)
  n=1
  *result = 1;
```

(i)



Application Example (2): rdigitValue

Write a recursive function that returns the value of the *k*th digit (k>0) from the right of a non-negative integer *num*. For example, if num=1234567 and k=3, the function will return 5 and if num=1234 and k=8, the function will return 0.

Write the function in two versions.

(1)The function **rdigitValue1()** returns the result, while

(2)rdigitValue2() passes the result through the parameter *result*.

The prototypes of the function are given below:

```
        int rdigitValue1(int num, int k);
           // by returning value
        void rdigitValue2(int num, int k, int *result);
           // by call by reference
```

Write a C program to test the functions.

```
Enter a number:
1284567
Enter the digit position:
3
rdigitValue1(): 5
rdigitValue2(): 5
```

Note:

When dealing with numbers, the integer division operator and modulus operator can be used to extract the digit value from the number.

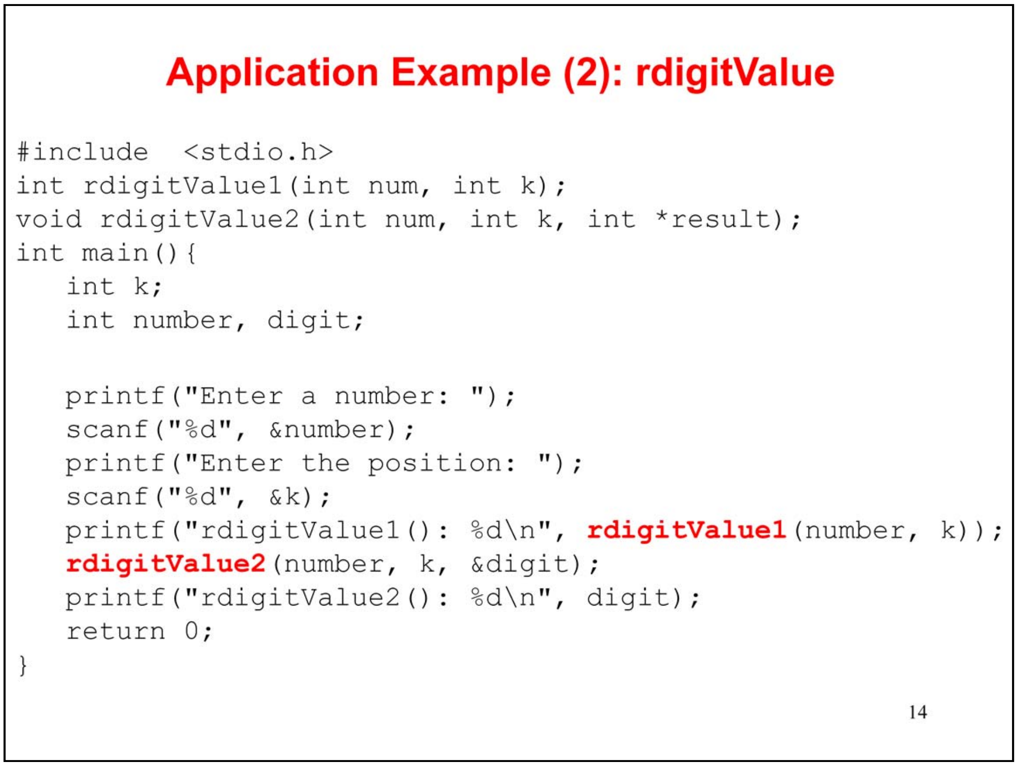
13

<u>Application Example (2) – rdigitValue</u>

1. The function prototypes are given below:

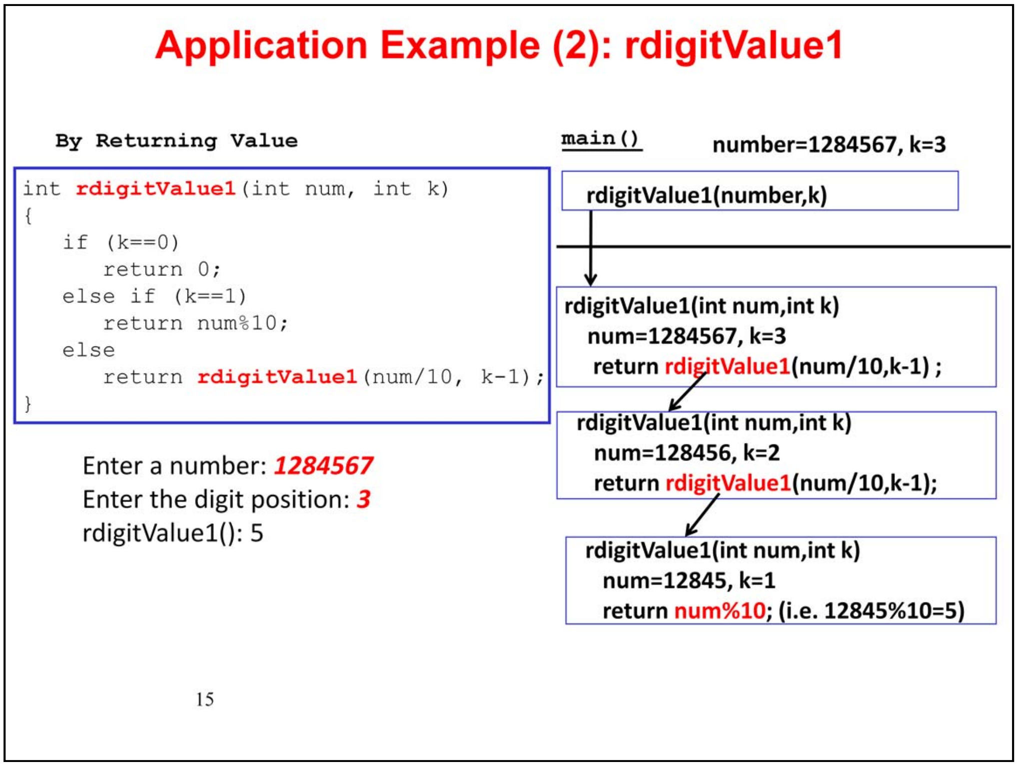
   int **rdigitValue1(**int num, int k);

   void **rdigitValue2(**int num, int k, int *result);



Application Example (2): rdigitValue

```
#include  <stdio.h>
int rdigitValue1(int num, int k);
void rdigitValue2(int num, int k, int *result);
int main(){
   int k;
   int number, digit;

   printf("Enter a number: ");
   scanf("%d", &number);
   printf("Enter the position: ");
   scanf("%d", &k);
   printf("rdigitValue1(): %d\n", rdigitValue1(number, k));
   rdigitValue2(number, k, &digit);
   printf("rdigitValue2(): %d\n", digit);
   return 0;
}
```



Application Example (2): rdigitValue1

By Returning Value

```
int rdigitValue1(int num, int k)
{
   if (k==0)
      return 0;
   else if (k==1)
      return num%10;
   else
      return rdigitValue1(num/10, k-1);
}
```

Enter a number: ***1284567***
Enter the digit position: ***3***
rdigitValue1(): 5

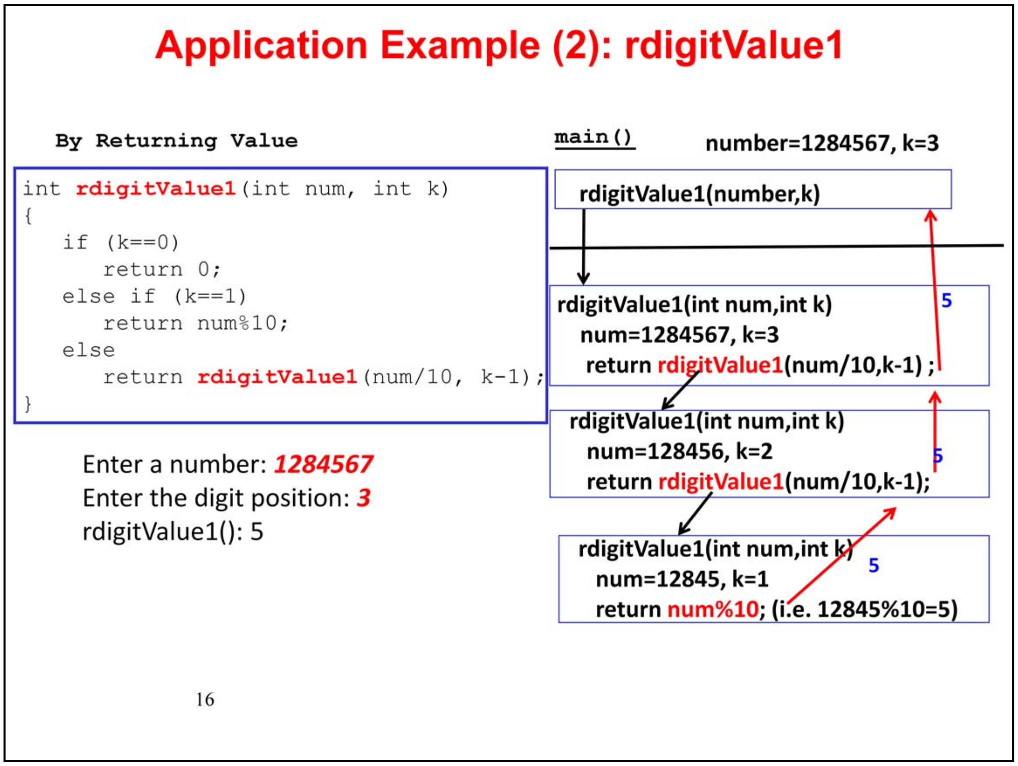
main() **number=1284567, k=3**

rdigitValue1(number,k)

rdigitValue1(int num,int k)
num=1284567, k=3
return rdigitValue1(num/10,k-1) ;

rdigitValue1(int num,int k)
num=128456, k=2
return rdigitValue1(num/10,k-1);

rdigitValue1(int num,int k)
num=12845, k=1
return num%10; (i.e. 12845%10=5)



Application Example (2): rdigitValue1

By Returning Value

```
int rdigitValue1(int num, int k)
{
   if (k==0)
      return 0;
   else if (k==1)
      return num%10;
   else
      return rdigitValue1(num/10, k-1);
}
```

```
Enter a number: 1284567
Enter the digit position: 3
rdigitValue1(): 5
```

main()

number=1284567, k=3

rdigitValue1(number,k)

rdigitValue1(int num,int k)
num=1284567, k=3
return rdigitValue1(num/10,k-1) ;

5

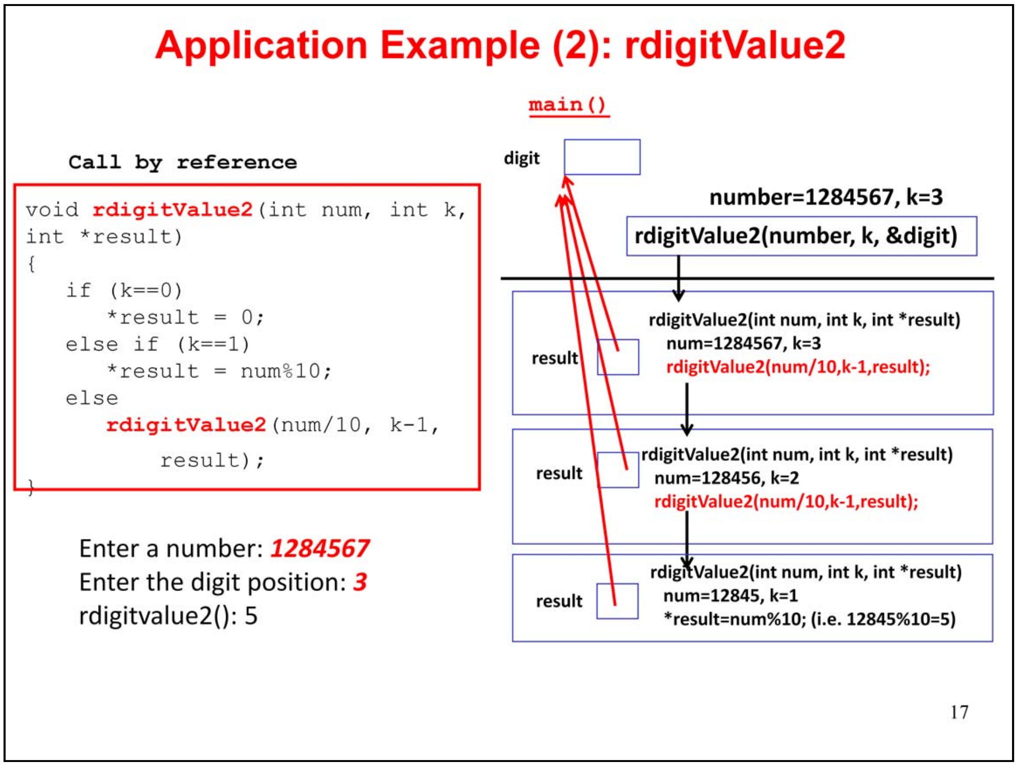
rdigitValue1(int num,int k)
num=128456, k=2
return rdigitValue1(num/10,k-1);

5

rdigitValue1(int num,int k)
num=12845, k=1
return num%10; (i.e. 12845%10=5)

5

16



Application Example (2): rdigitValue2

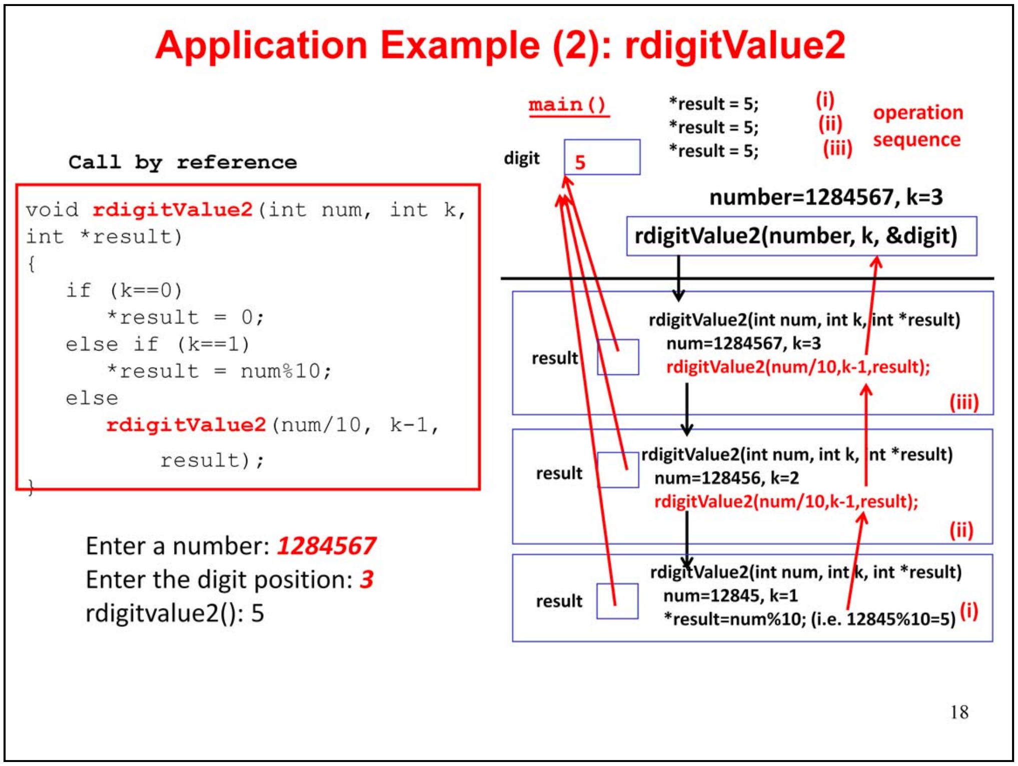
Call by reference

```
void rdigitValue2(int num, int k,
int *result)
{
   if (k==0)
      *result = 0;
   else if (k==1)
      *result = num%10;
   else
      rdigitValue2(num/10, k-1,
          result);
}
```

Enter a number: ***1284567***
Enter the digit position: ***3***
rdigitvalue2(): 5

main()
digit
number=1284567, k=3
rdigitValue2(number, k, &digit)
result
rdigitValue2(int num, int k, int *result)
num=1284567, k=3
rdigitValue2(num/10,k-1,result);
result
rdigitValue2(int num, int k, int *result)
num=128456, k=2
rdigitValue2(num/10,k-1,result);
result
rdigitValue2(int num, int k, int *result)
num=12845, k=1
*result=num%10; (i.e. 12845%10=5)

17



Application Example (2): rdigitValue2

Call by reference

```
void rdigitValue2(int num, int k,
int *result)
{
   if (k==0)
      *result = 0;
   else if (k==1)
      *result = num%10;
   else
      rdigitValue2(num/10, k-1,
          result);
}
```

```
Enter a number: 1284567
Enter the digit position: 3
rdigitvalue2(): 5
```

main()

digit 5

*result = 5; (i)
*result = 5; (ii)
*result = 5; (iii)

operation sequence

number=1284567, k=3

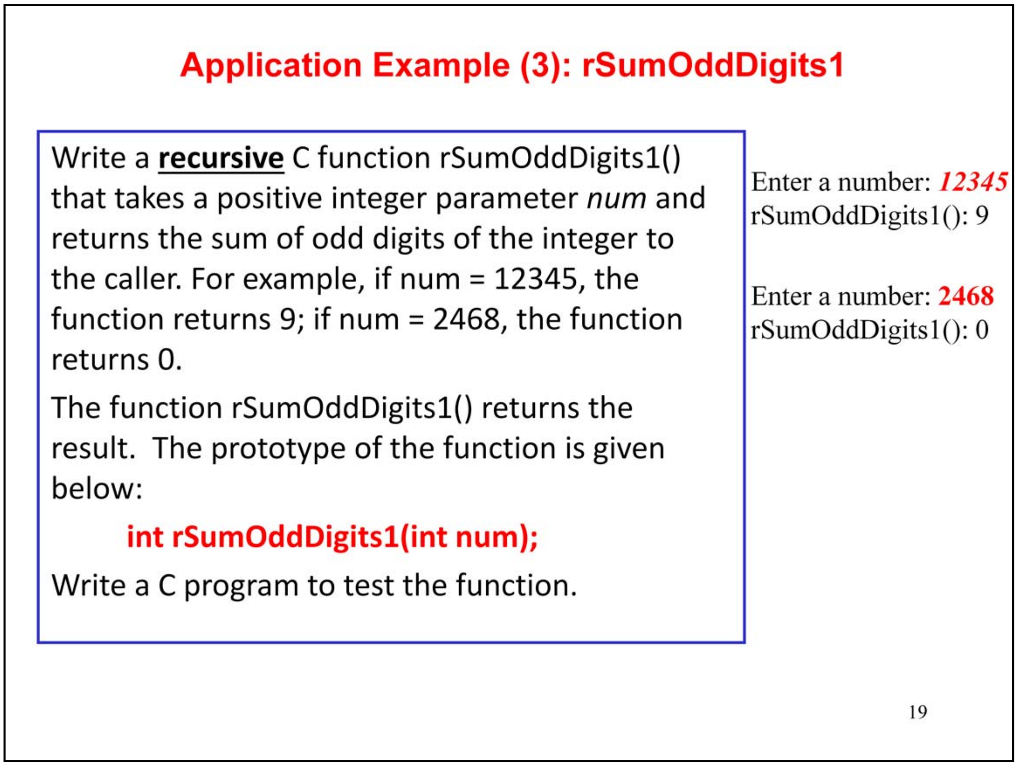
rdigitValue2(number, k, &digit)

result
rdigitValue2(int num, int k, int *result)
num=1284567, k=3
rdigitValue2(num/10,k-1,result);
(iii)

result
rdigitValue2(int num, int k, int *result)
num=128456, k=2
rdigitValue2(num/10,k-1,result);
(ii)

result
rdigitValue2(int num, int k, int *result)
num=12845, k=1
*result=num%10; (i.e. 12845%10=5) (i)

18



Application Example (3): rSumOddDigits1

Write a **recursive** C function rSumOddDigits1() that takes a positive integer parameter *num* and returns the sum of odd digits of the integer to the caller. For example, if num = 12345, the function returns 9; if num = 2468, the function returns 0.

The function rSumOddDigits1() returns the result. The prototype of the function is given below:

int rSumOddDigits1(int num);

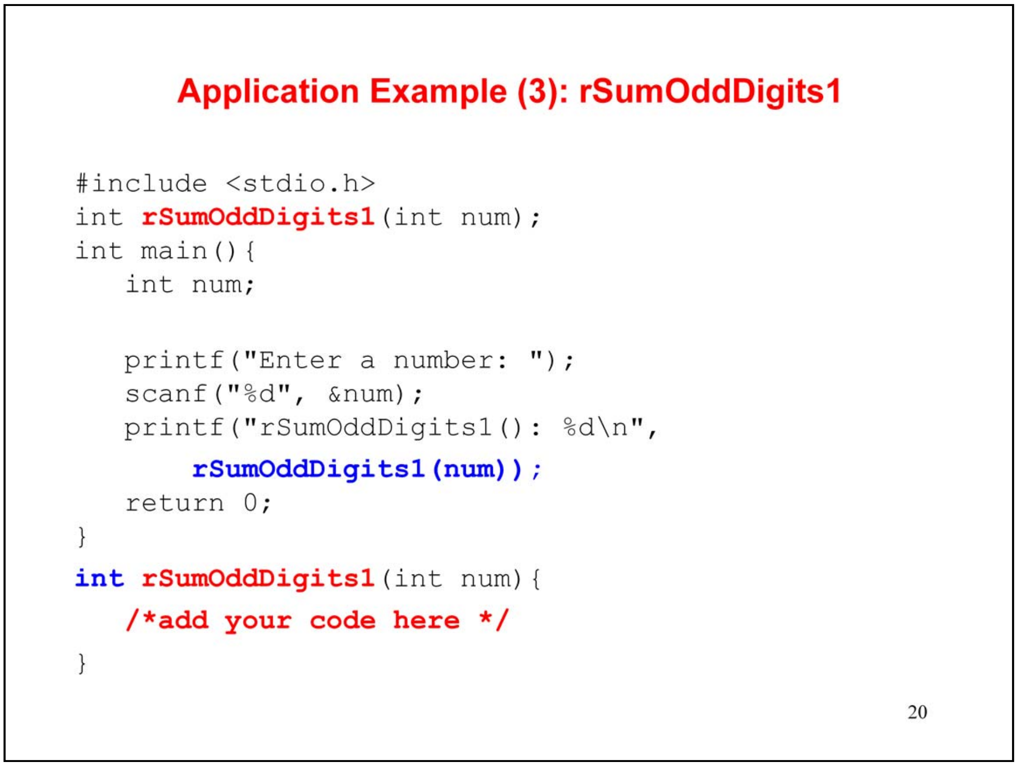
Write a C program to test the function.

```
Enter a number: 12345
rSumOddDigits1(): 9

Enter a number: 2468
rSumOddDigits1(): 0
```

Application Example (3) – rSumOddDigits1

1. The function prototype is given below:

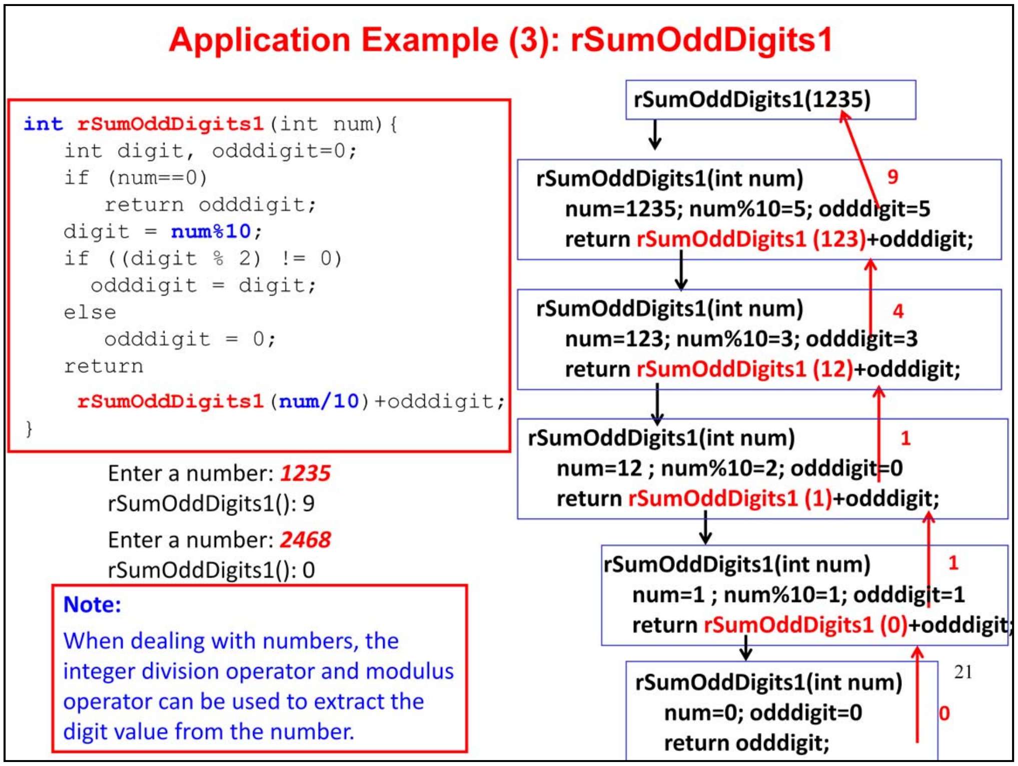
    int **rSumOddDigits1(**int num);



Application Example (3): rSumOddDigits1

```
#include <stdio.h>
int rSumOddDigits1(int num);
int main(){
   int num;

   printf("Enter a number: ");
   scanf("%d", &num);
   printf("rSumOddDigits1(): %d\n",
       rSumOddDigits1(num));
   return 0;
}
int rSumOddDigits1(int num){
   /*add your code here */
}
```

Vcbv



Application Example (3): rSumOddDigits1

```
int rSumOddDigits1(int num){
   int digit, odddigit=0;
   if (num==0)
      return odddigit;
   digit = num%10;
   if ((digit % 2) != 0)
     odddigit = digit;
   else
      odddigit = 0;
   return
    rSumOddDigits1(num/10)+odddigit;
}
```

```
Enter a number: 1235
rSumOddDigits1(): 9

Enter a number: 2468
rSumOddDigits1(): 0
```

Note:

When dealing with numbers, the integer division operator and modulus operator can be used to extract the digit value from the number.

rSumOddDigits1(1235)

rSumOddDigits1(int num)
num=1235; num%10=5; odddigit=5
return rSumOddDigits1 (123)+odddigit;

9

rSumOddDigits1(int num)
num=123; num%10=3; odddigit=3
return rSumOddDigits1 (12)+odddigit;

4

rSumOddDigits1(int num)
num=12 ; num%10=2; odddigit=0
return rSumOddDigits1 (1)+odddigit;

1

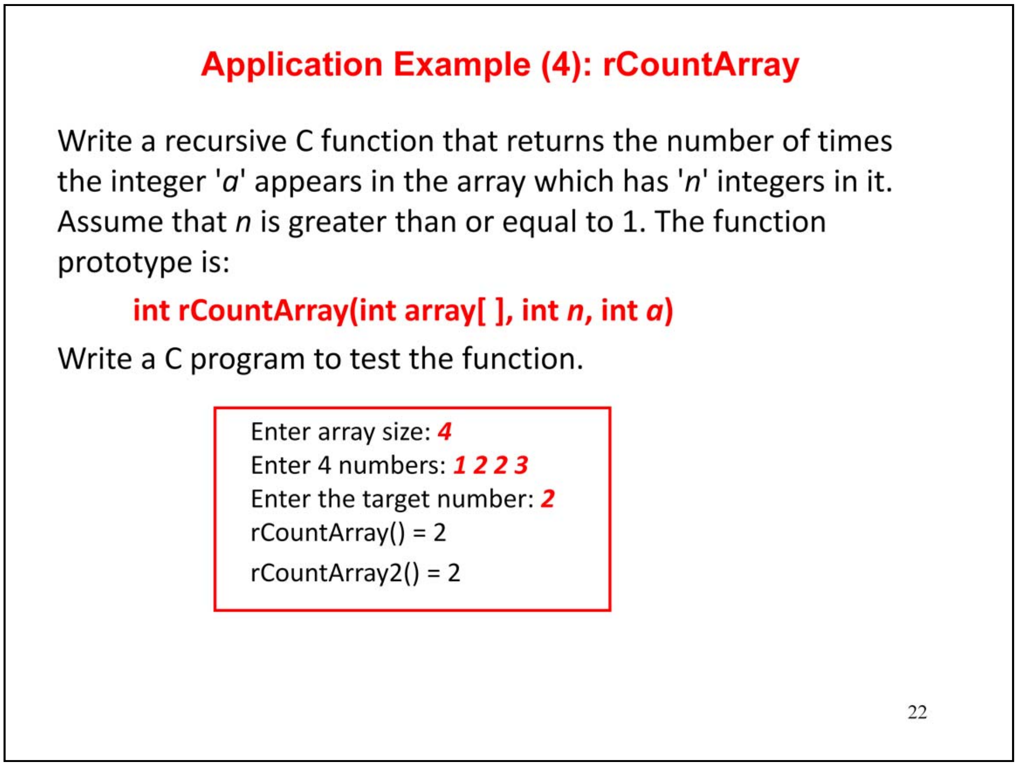
rSumOddDigits1(int num)
num=1 ; num%10=1; odddigit=1
return rSumOddDigits1 (0)+odddigit;

1

rSumOddDigits1(int num)
num=0; odddigit=0
return odddigit;

0

21



Application Example (4): rCountArray

Write a recursive C function that returns the number of times the integer '*a*' appears in the array which has '*n*' integers in it. Assume that *n* is greater than or equal to 1. The function prototype is:

int rCountArray(int array[ ], int *n*, int *a*)

Write a C program to test the function.

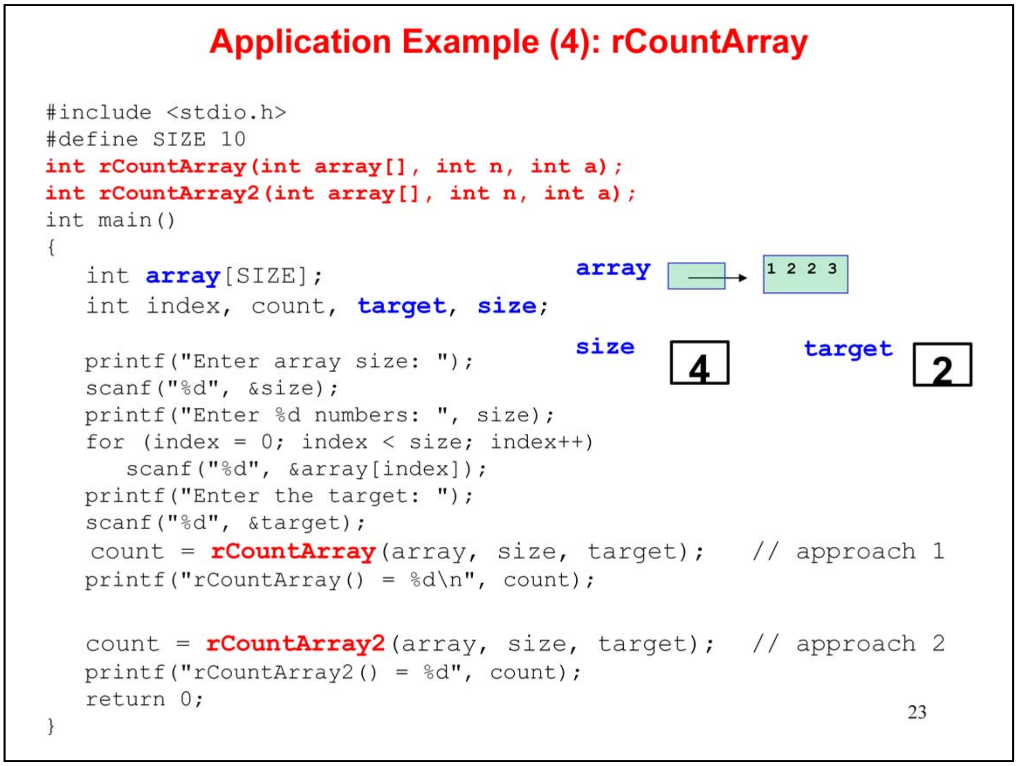
```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray() = 2
rCountArray2() = 2
```

Application Example (4) – rCountArray

1. The function prototypes are given below:

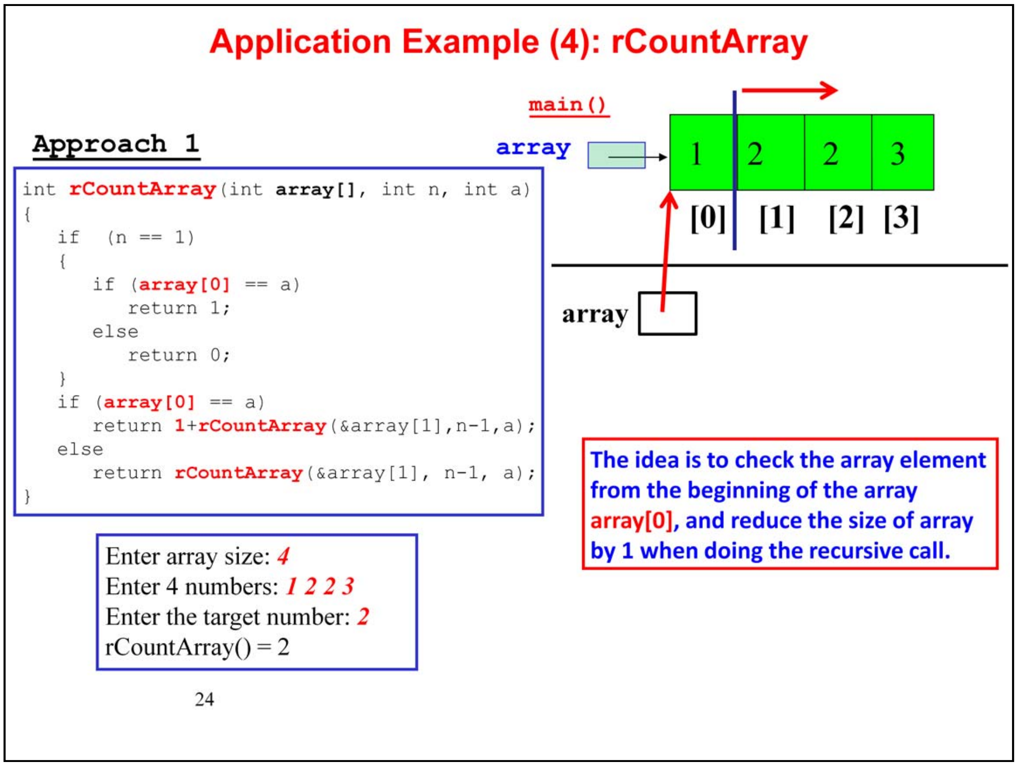
   int **rCountArray(**int array[], int n, int a);

   int **rCountArray2(**int array[], int n, int a);


array
1 2 2 3
size
4
target
2

Application Example (4): rCountArray

```
#include <stdio.h>
#define SIZE 10
int rCountArray(int array[], int n, int a);
int rCountArray2(int array[], int n, int a);
int main()
{
   int array[SIZE];
   int index, count, target, size;

   printf("Enter array size: ");
   scanf("%d", &size);
   printf("Enter %d numbers: ", size);
   for (index = 0; index < size; index++)
      scanf("%d", &array[index]);
   printf("Enter the target: ");
   scanf("%d", &target);
    count = rCountArray(array, size, target);   // approach 1
   printf("rCountArray() = %d\n", count);

   count = rCountArray2(array, size, target);   // approach 2
   printf("rCountArray2() = %d", count);
   return 0;
}
```



Application Example (4): rCountArray

Approach 1

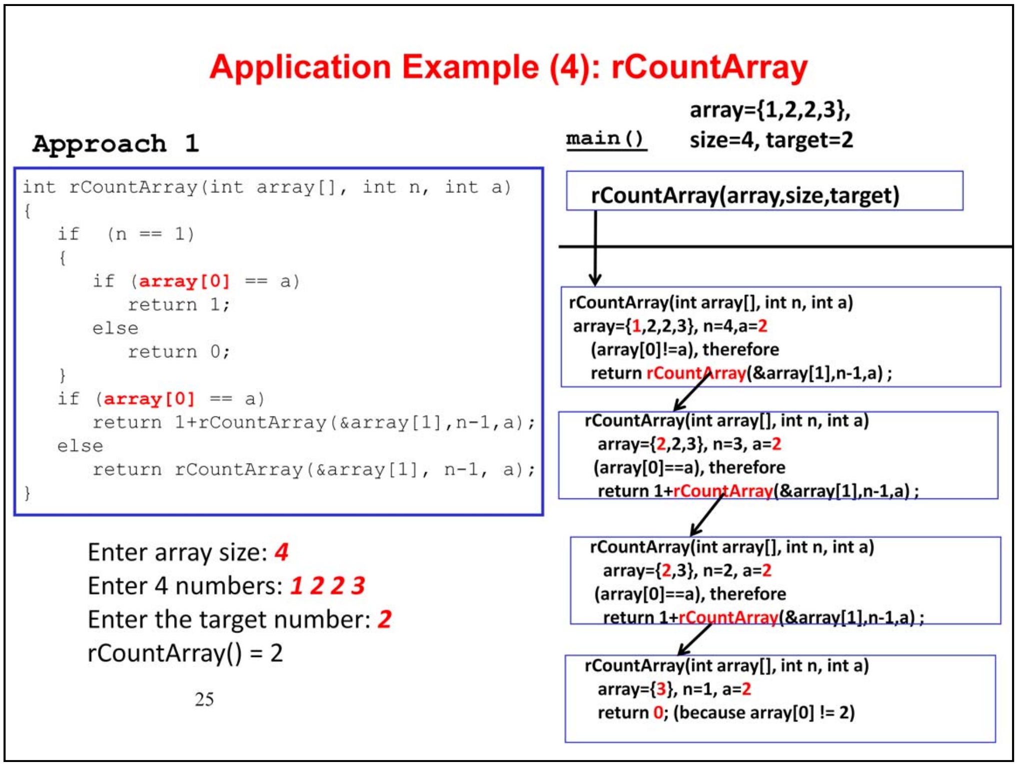
```
int rCountArray(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[0] == a)
      return 1+rCountArray(&array[1],n-1,a);
   else
      return rCountArray(&array[1], n-1, a);
}
```

main()
array
1
2
2
3
[0]
[1]
[2]
[3]
array

The idea is to check the array element from the beginning of the array array[0], and reduce the size of array by 1 when doing the recursive call.

```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray() = 2
```

24



Application Example (4): rCountArray

Approach 1

```
int rCountArray(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[0] == a)
      return 1+rCountArray(&array[1],n-1,a);
   else
      return rCountArray(&array[1], n-1, a);
}
```

```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray() = 2
```

main() array={1,2,2,3}, size=4, target=2

rCountArray(array,size,target)

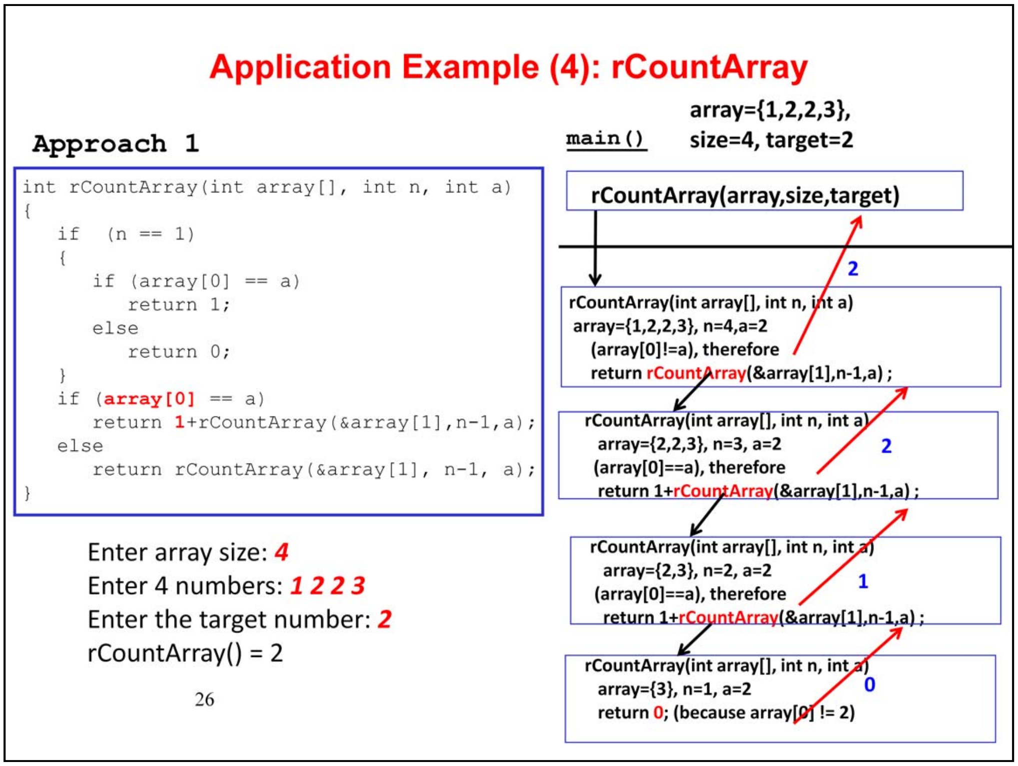
rCountArray(int array[], int n, int a)
array={1,2,2,3}, n=4,a=2
(array[0]!=a), therefore
return rCountArray(&array[1],n-1,a) ;

rCountArray(int array[], int n, int a)
array={2,2,3}, n=3, a=2
(array[0]==a), therefore
return 1+rCountArray(&array[1],n-1,a) ;

rCountArray(int array[], int n, int a)
array={2,3}, n=2, a=2
(array[0]==a), therefore
return 1+rCountArray(&array[1],n-1,a) ;

rCountArray(int array[], int n, int a)
array={3}, n=1, a=2
return 0; (because array[0] != 2)

25



Application Example (4): rCountArray

Approach 1

```
int rCountArray(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[0] == a)
      return 1+rCountArray(&array[1],n-1,a);
   else
      return rCountArray(&array[1], n-1, a);
}
```

```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray() = 2
```

main()

array={1,2,2,3}, size=4, target=2

rCountArray(array,size,target)

2

rCountArray(int array[], int n, int a)
array={1,2,2,3}, n=4,a=2
(array[0]!=a), therefore
return rCountArray(&array[1],n-1,a) ;

2

rCountArray(int array[], int n, int a)
array={2,2,3}, n=3, a=2
(array[0]==a), therefore
return 1+rCountArray(&array[1],n-1,a) ;

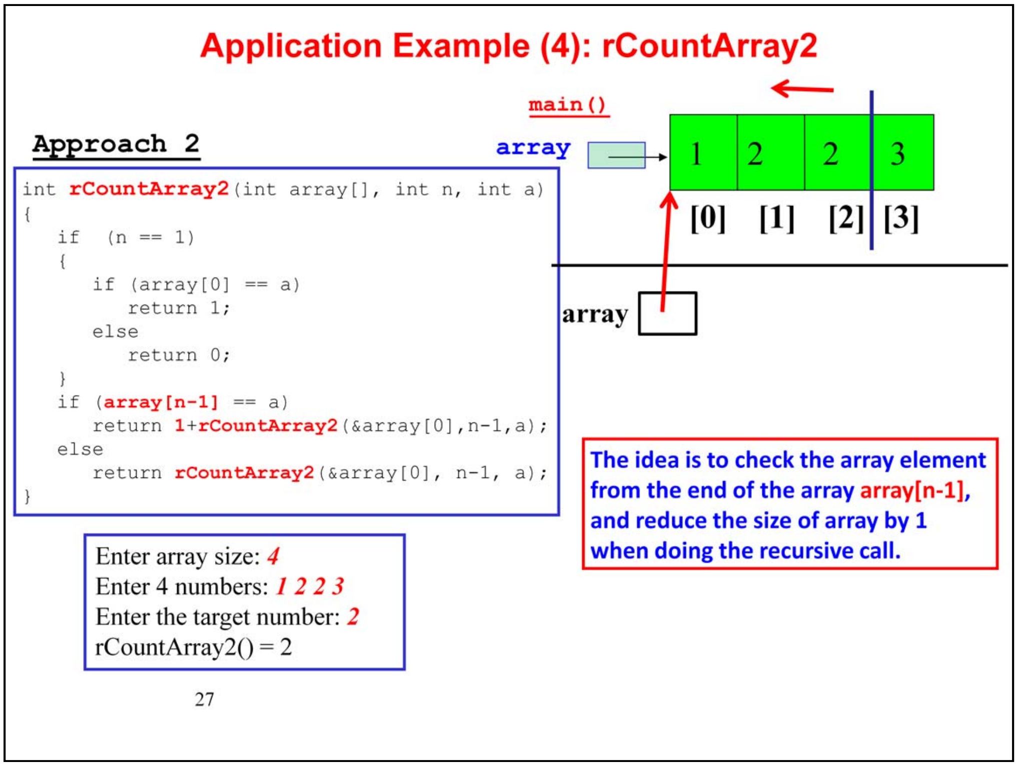
1

rCountArray(int array[], int n, int a)
array={2,3}, n=2, a=2
(array[0]==a), therefore
return 1+rCountArray(&array[1],n-1,a) ;

0

rCountArray(int array[], int n, int a)
array={3}, n=1, a=2
return 0; (because array[0] != 2)

26



Application Example (4): rCountArray2

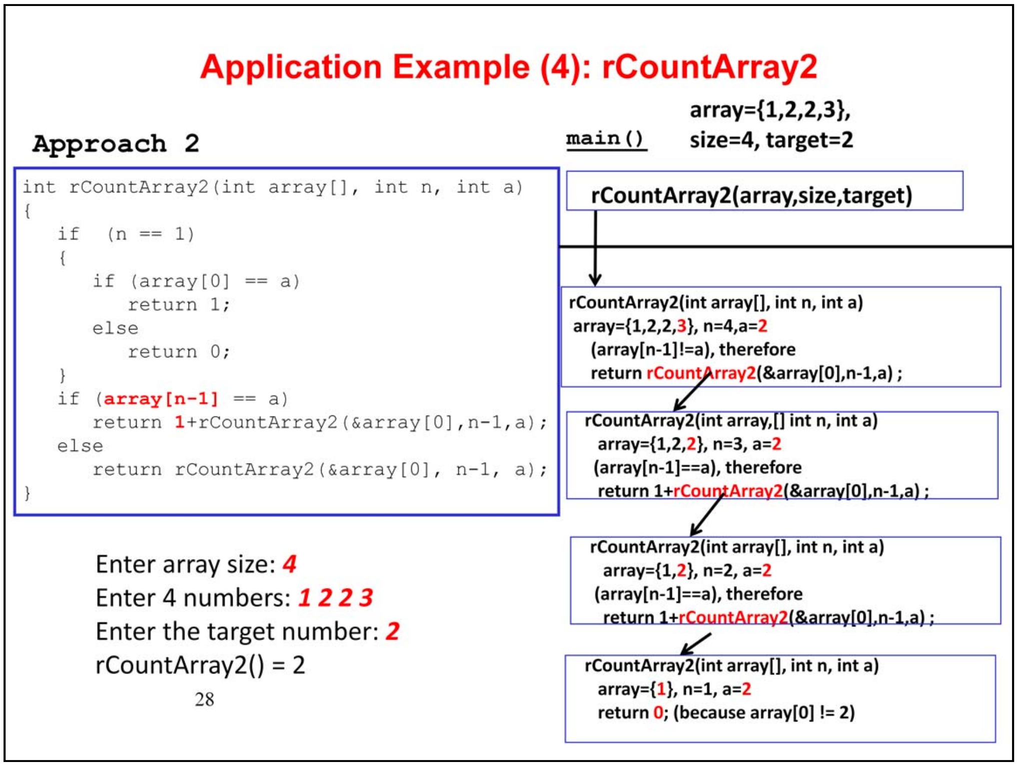
main()
array
1
2
2
3
[0]
[1]
[2]
[3]
array

Approach 2

```
int rCountArray2(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[n-1] == a)
      return 1+rCountArray2(&array[0],n-1,a);
   else
      return rCountArray2(&array[0], n-1, a);
}
```

The idea is to check the array element from the end of the array array[n-1], and reduce the size of array by 1 when doing the recursive call.

```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray2() = 2
```



Application Example (4): rCountArray2

Approach 2

```
int rCountArray2(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[n-1] == a)
      return 1+rCountArray2(&array[0],n-1,a);
   else
      return rCountArray2(&array[0], n-1, a);
}
```

```
Enter array size: 4
Enter 4 numbers: 1 2 2 3
Enter the target number: 2
rCountArray2() = 2
```

28

main() array={1,2,2,3}, size=4, target=2

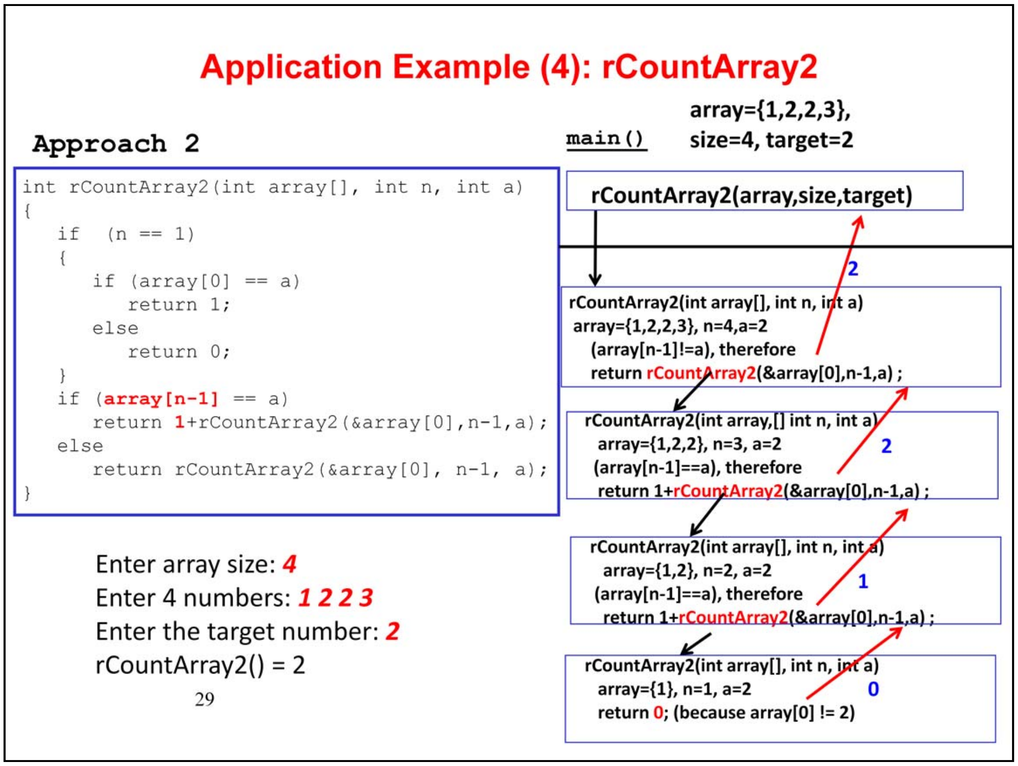
rCountArray2(array,size,target)

rCountArray2(int array[], int n, int a)
array={1,2,2,3}, n=4,a=2
(array[n-1]!=a), therefore
return rCountArray2(&array[0],n-1,a) ;

rCountArray2(int array,[] int n, int a)
array={1,2,2}, n=3, a=2
(array[n-1]==a), therefore
return 1+rCountArray2(&array[0],n-1,a) ;

rCountArray2(int array[], int n, int a)
array={1,2}, n=2, a=2
(array[n-1]==a), therefore
return 1+rCountArray2(&array[0],n-1,a) ;

rCountArray2(int array[], int n, int a)
array={1}, n=1, a=2
return 0; (because array[0] != 2)



Application Example (4): rCountArray2

Approach 2

```
int rCountArray2(int array[], int n, int a)
{
   if  (n == 1)
   {
      if (array[0] == a)
         return 1;
      else
         return 0;
   }
   if (array[n-1] == a)
      return 1+rCountArray2(&array[0],n-1,a);
   else
      return rCountArray2(&array[0], n-1, a);
}
```

main() **array={1,2,2,3},**
size=4, target=2

rCountArray2(array,size,target)

rCountArray2(int array[], int n, int a)
array={1,2,2,3}, n=4,a=2
(array[n-1]!=a), therefore
return rCountArray2(&array[0],n-1,a) ;

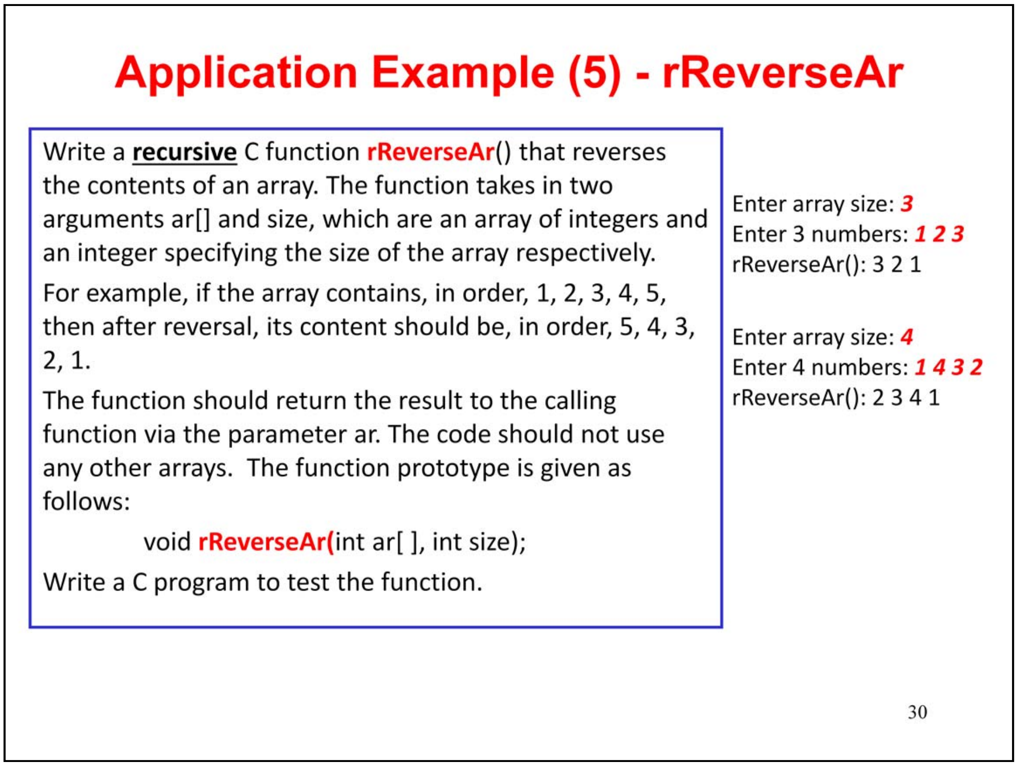
rCountArray2(int array,[] int n, int a)
array={1,2,2}, n=3, a=2
(array[n-1]==a), therefore
return 1+rCountArray2(&array[0],n-1,a) ;

rCountArray2(int array[], int n, int a)
array={1,2}, n=2, a=2
(array[n-1]==a), therefore
return 1+rCountArray2(&array[0],n-1,a) ;

rCountArray2(int array[], int n, int a)
array={1}, n=1, a=2
return 0; (because array[0] != 2)

Enter array size: ***4***
Enter 4 numbers: ***1 2 2 3***
Enter the target number: ***2***
rCountArray2() = 2

29



Application Example (5) - rReverseAr

Write a **recursive** C function **rReverseAr**() that reverses the contents of an array. The function takes in two arguments ar[] and size, which are an array of integers and an integer specifying the size of the array respectively.

For example, if the array contains, in order, 1, 2, 3, 4, 5, then after reversal, its content should be, in order, 5, 4, 3, 2, 1.

The function should return the result to the calling function via the parameter ar. The code should not use any other arrays. The function prototype is given as follows:

void **rReverseAr(**int ar[ ], int size);

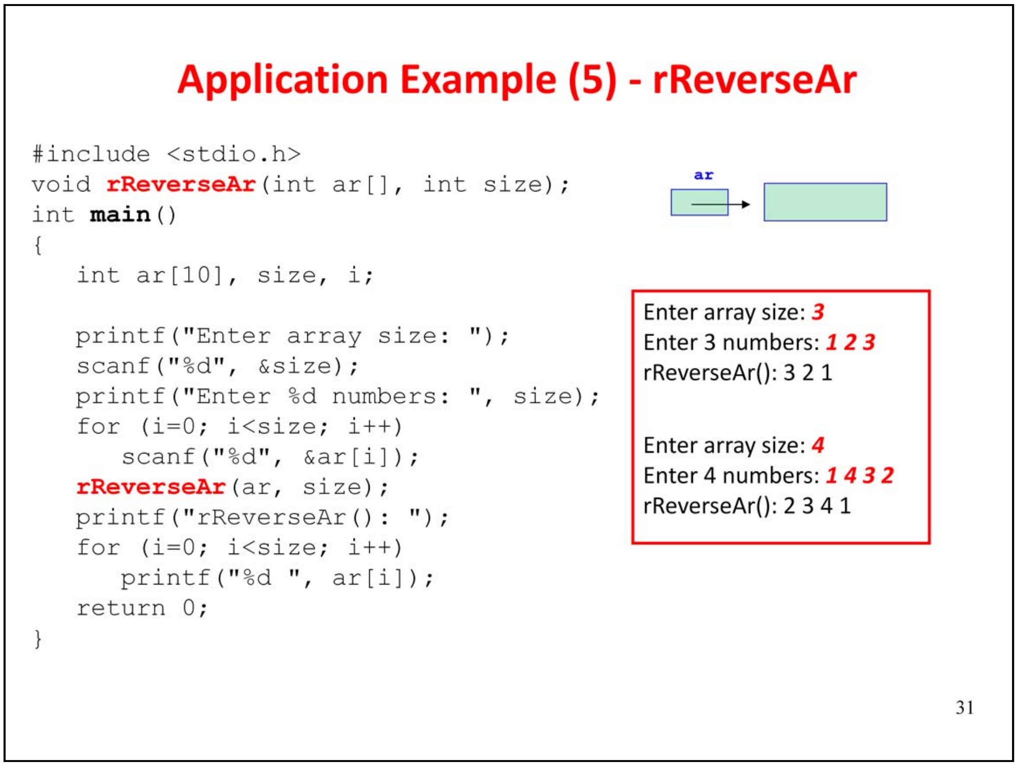
Write a C program to test the function.

```
Enter array size: 3
Enter 3 numbers: 1 2 3
rReverseAr(): 3 2 1

Enter array size: 4
Enter 4 numbers: 1 4 3 2
rReverseAr(): 2 3 4 1
```

Application Example (5) – rReverseAr

1. The application aims to reverse the contents in an array. The function prototype is given below:

   void **rReverseAr(**int ar[ ], int size);



Application Example (5) - rReverseAr

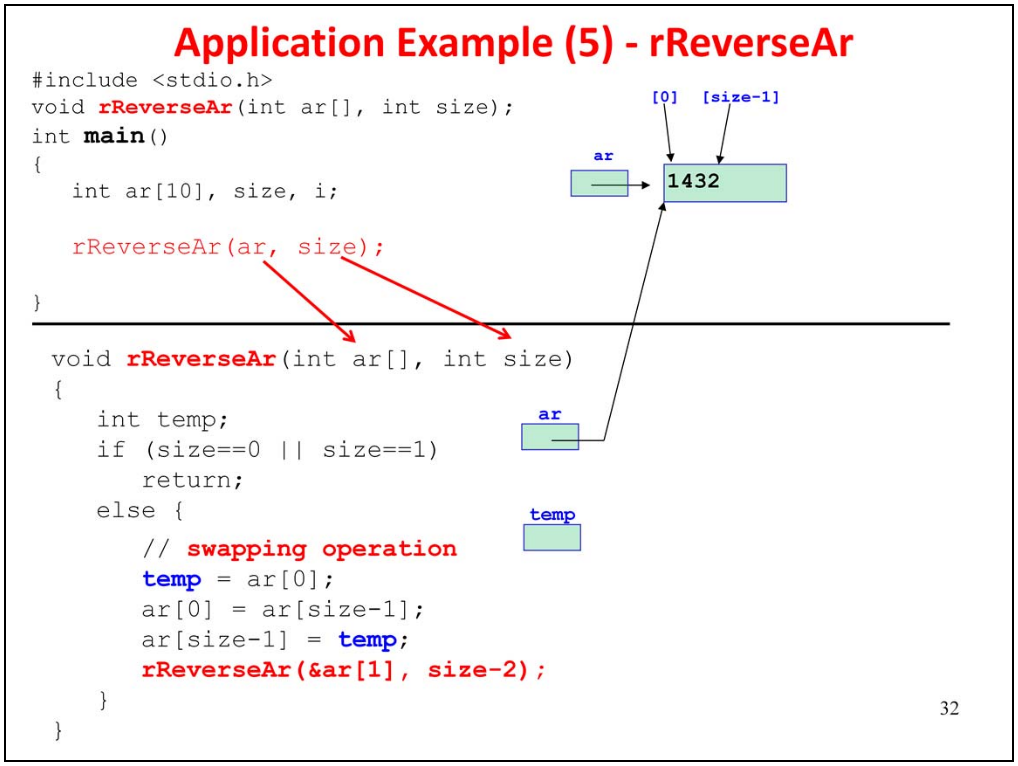
```
#include <stdio.h>
void rReverseAr(int ar[], int size);
int main()
{
   int ar[10], size, i;

   printf("Enter array size: ");
   scanf("%d", &size);
   printf("Enter %d numbers: ", size);
   for (i=0; i<size; i++)
      scanf("%d", &ar[i]);
   rReverseAr(ar, size);
   printf("rReverseAr(): ");
   for (i=0; i<size; i++)
      printf("%d ", ar[i]);
   return 0;
}
```

```
Enter array size: 3
Enter 3 numbers: 1 2 3
rReverseAr(): 3 2 1

Enter array size: 4
Enter 4 numbers: 1 4 3 2
rReverseAr(): 2 3 4 1
```

31

Application Example (5) – rReverseAr

1. The main function is given. It calls the recursive function rReverseAr() after reading in user input data.



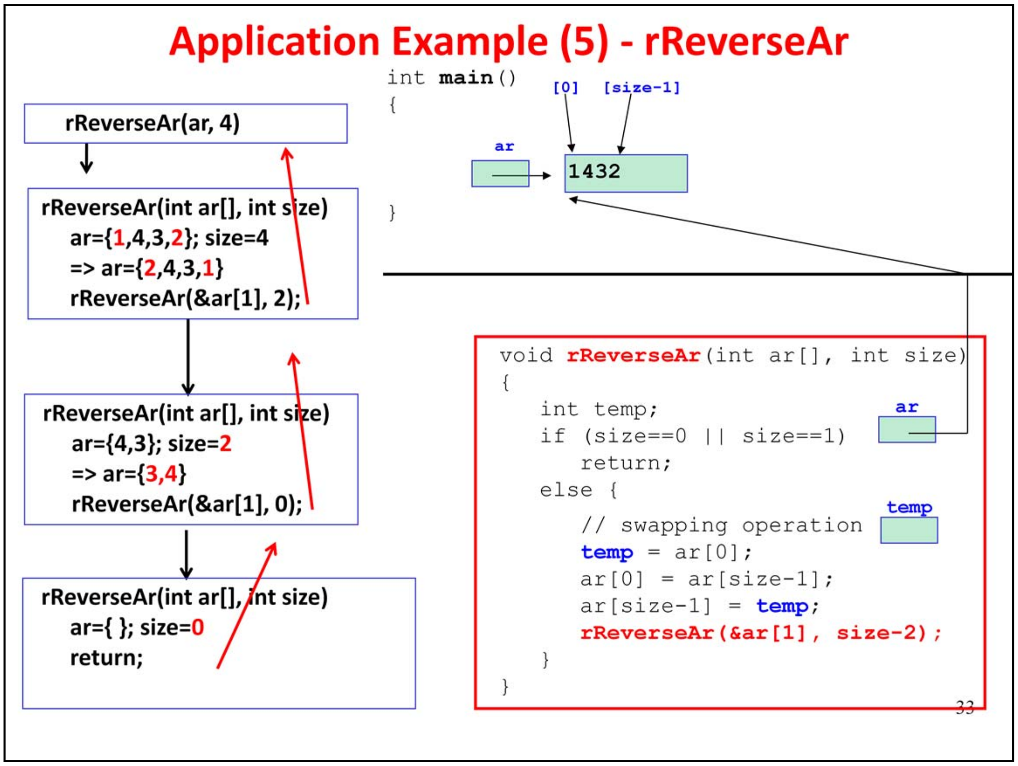
Application Example (5) - rReverseAr

```
#include <stdio.h>
void rReverseAr(int ar[], int size);
int main()
{
   int ar[10], size, i;

   rReverseAr(ar, size);

}
```

```
 void rReverseAr(int ar[], int size)
 {
    int temp;
    if (size==0 || size==1)
       return;
    else {
       // swapping operation
       temp = ar[0];
       ar[0] = ar[size-1];
       ar[size-1] = temp;
       rReverseAr(&ar[1], size-2);
    }
 }
```

Application Example (5) – rReverseAr

1. The **rReverseAr()** function is given.
2. The terminating condition happens when the size of the array is either 0 or 1.
3. For the recursive condition, a swapping operation is performed between the first element and the last element of the array. The function is called recursively by passing in the address of the element with index 1 (i.e. &ar[1]), and size-2 as parameters.



Application Example (5) - rReverseAr

rReverseAr(ar, 4)

rReverseAr(int ar[], int size)
ar={1,4,3,2}; size=4
=> ar={2,4,3,1}
rReverseAr(&ar[1], 2);

rReverseAr(int ar[], int size)
ar={4,3}; size=2
=> ar={3,4}
rReverseAr(&ar[1], 0);

rReverseAr(int ar[], int size)
ar={ }; size=0
return;

```
int main()
{

}
```

```
void rReverseAr(int ar[], int size)
{
   int temp;
   if (size==0 || size==1)
      return;
   else {
      // swapping operation
      temp = ar[0];
      ar[0] = ar[size-1];
      ar[size-1] = temp;
      rReverseAr(&ar[1], size-2);
   }
}
```

33

Application Example (5) – rReverseAr

1. The tracing of the rReverseAr function is shown.