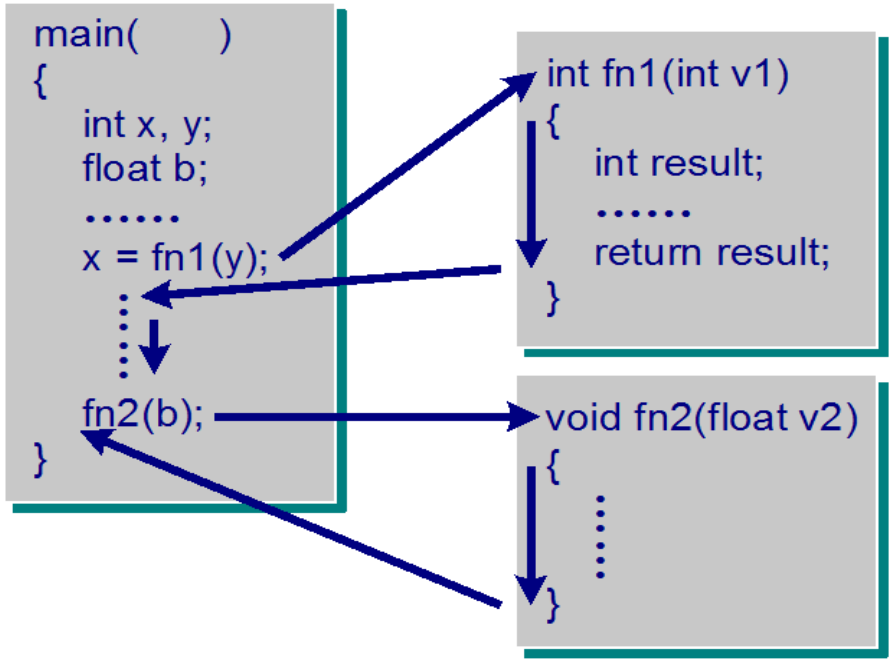
Tutorial 6 (Week 7)
Recursive Functions

Function Execution

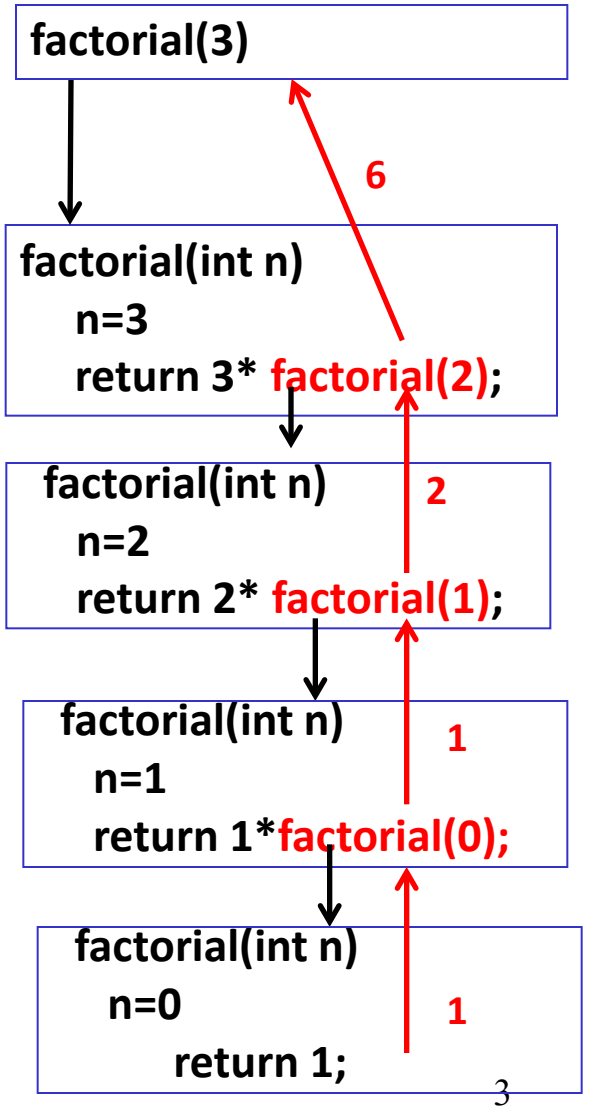
- C **Functions** (iterative) have the following properties:
  - A function, when called, will accomplish a certain job.
  - When a function **fn1()** is called, control is transferred from the calling point to the first statement in **fn1()**. After the function finishes execution, the control will 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.**



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);
}
```



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)

Q1 (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.

Q1 (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;
}
```

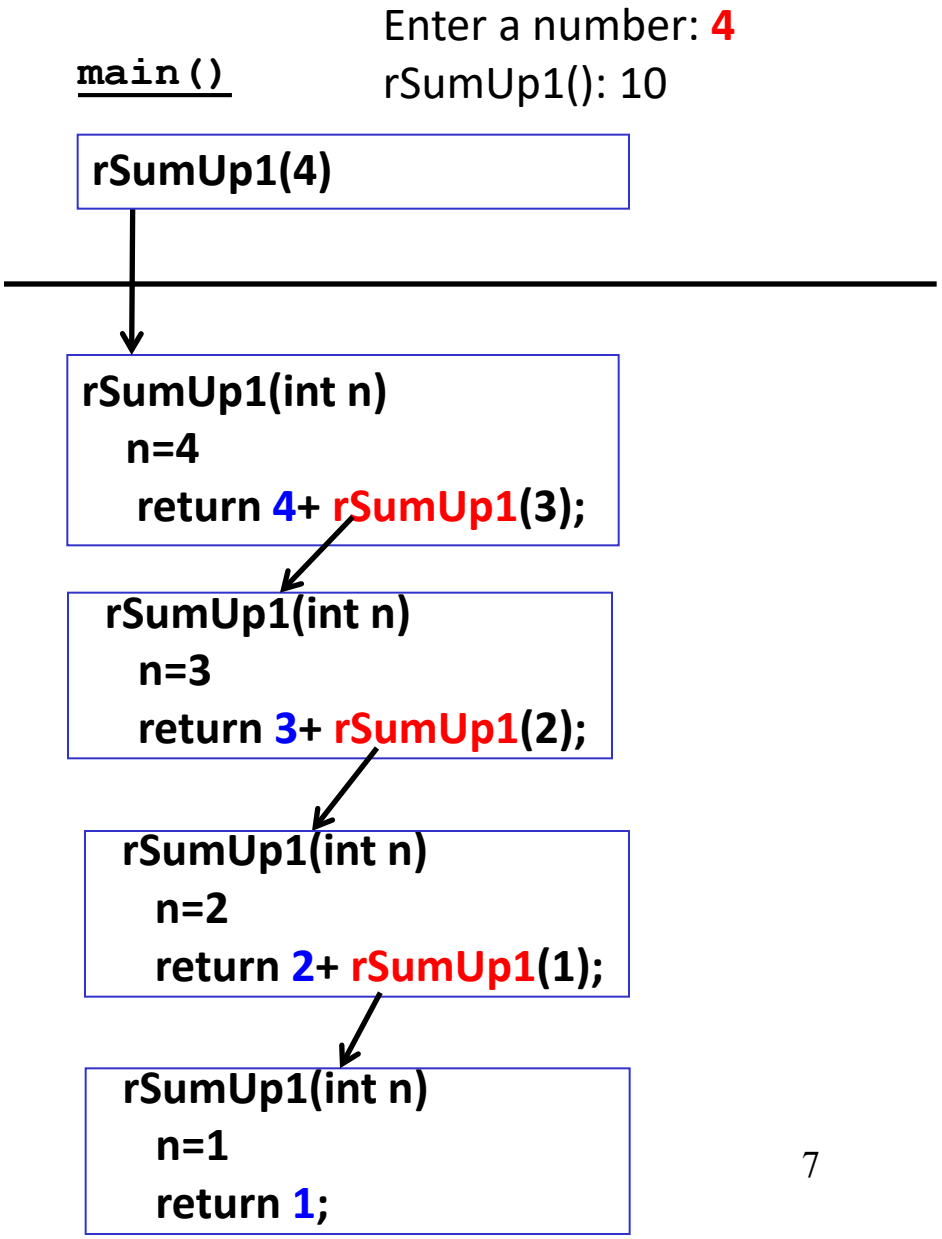
Q1 (rSumUp1)

By Returning Value

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

Enter a number: 4
rSumUp1(): 10

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



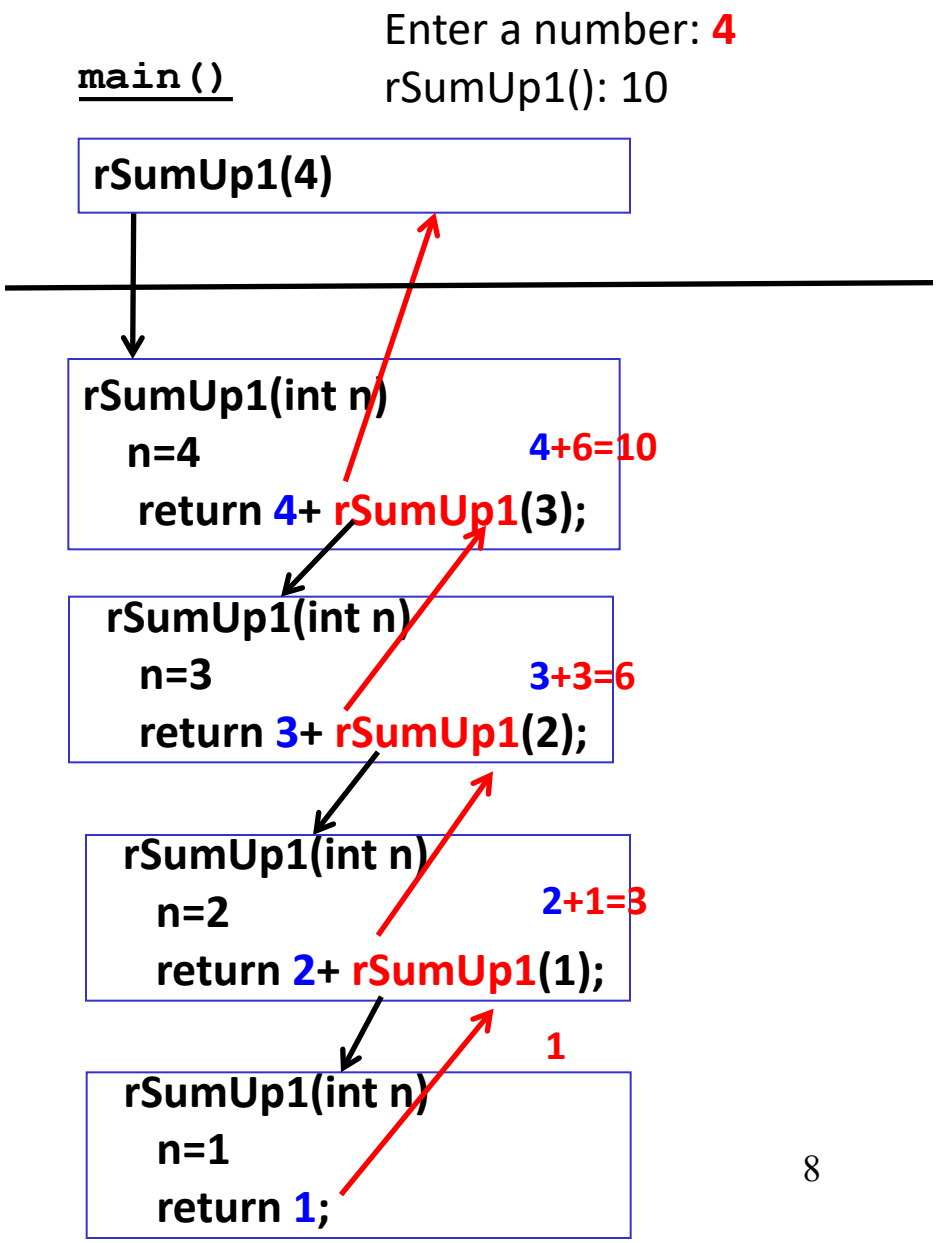
Q1 (rSumUp1)

By Returning Value

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

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

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



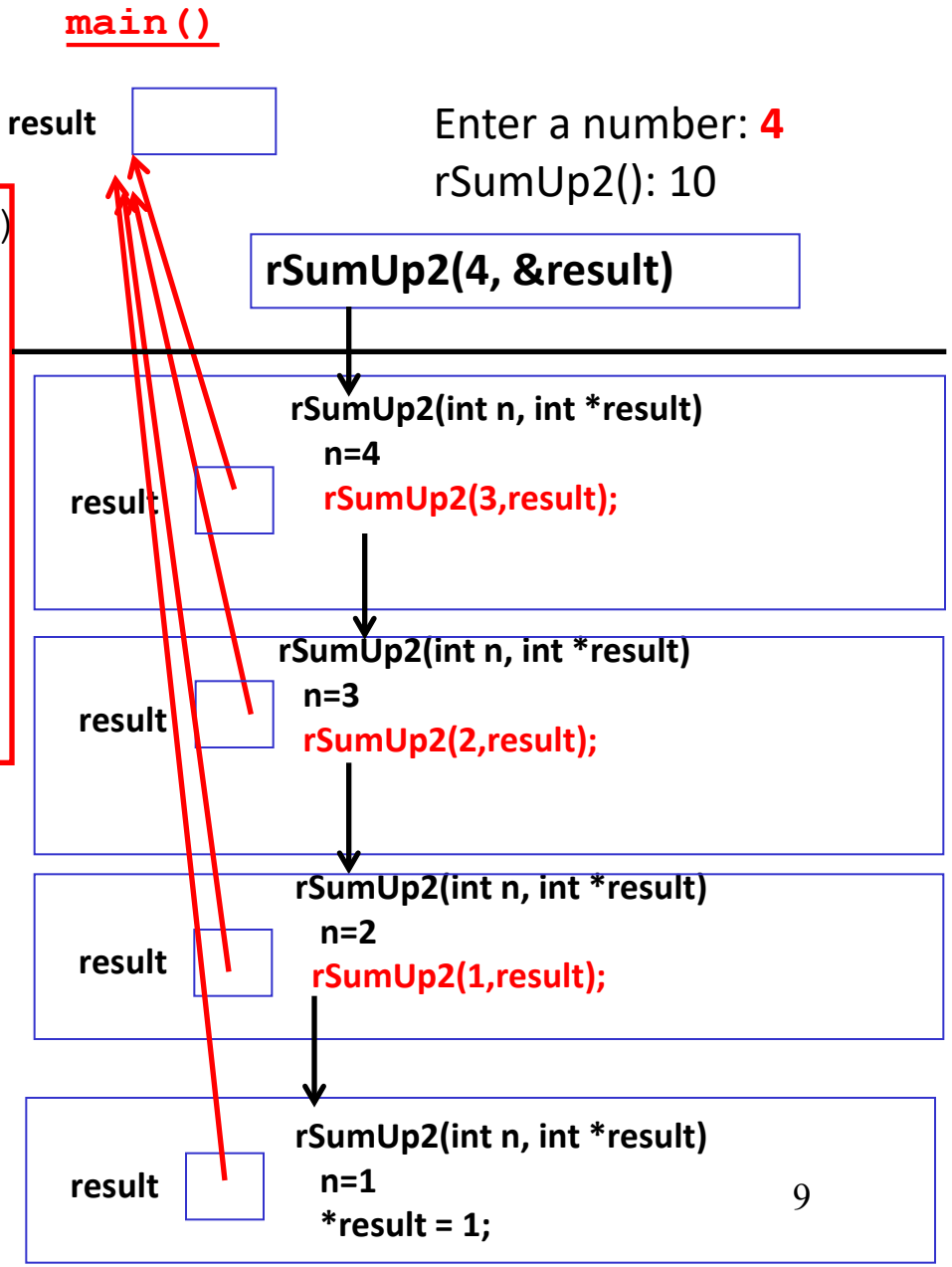
Q1 (rSumUp2)

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

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



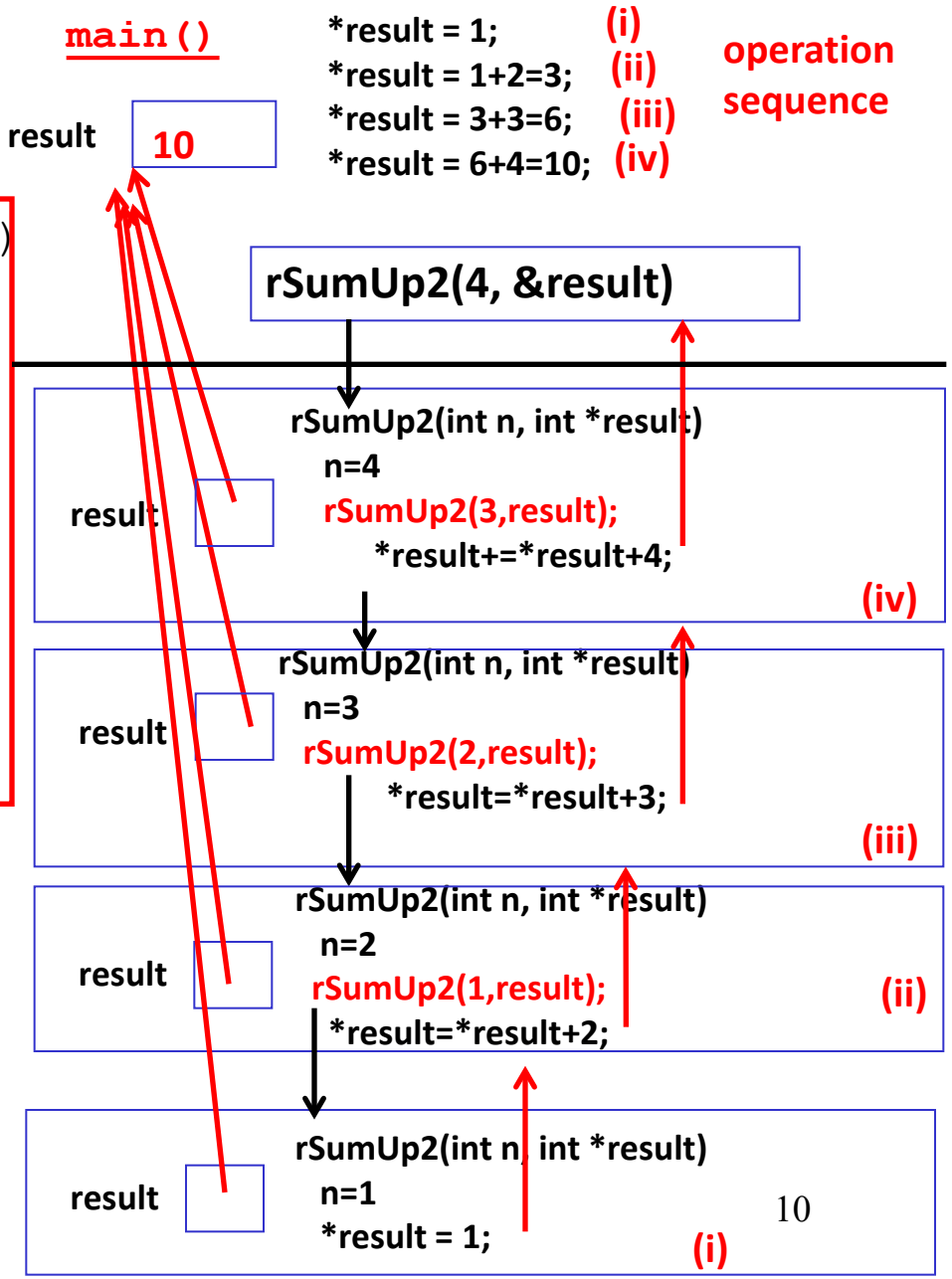
Q1 (rSumUp2)

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

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



Q2 (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 is12348567 and k is 3, the function will return 5 and if num is 1234 and k is 8, the function will return **0**.

Write the recursive function in two versions. The function **rDigitValue1()** computes and returns the result. The function **rDigitValue2()** computes and returns the result through the parameter result using call by reference. The function prototypes are given below:

```
int rDigitValue1(int num, int k);
void rDigitValue2(int num, int k, int *result);
```

Write a program to test the functions.

```
Enter a number:
2348567
Enter k position:
3
rDigitValue1(): 5
rDigitValue2(): 5

Enter a number:
123
Enter k position:
4
rDigitValue1(): 0
rDigitValue2(): 0
```

Q2 (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: \n");
   scanf("%d", &number);
   printf("Enter k position: \n");
   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);

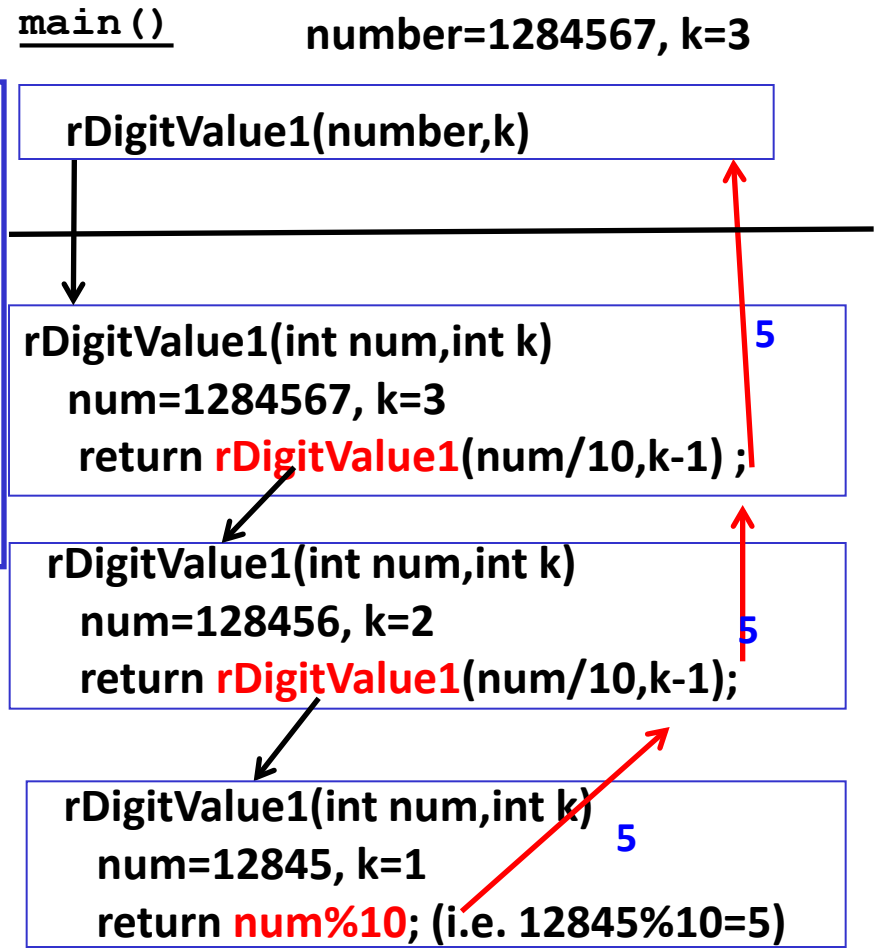
rDigitValue1(int num,int k)
num=12845, k=1
return num%10; (i.e. 12845%10=5)

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

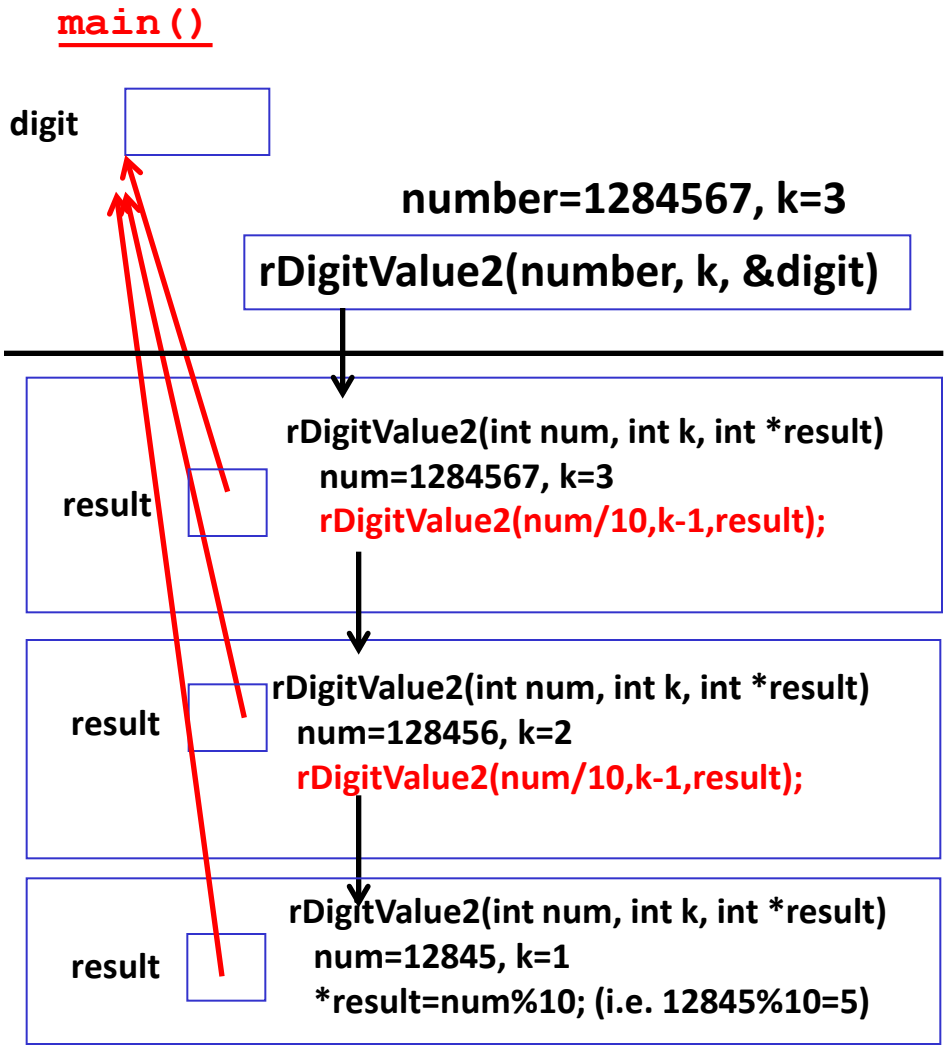


Application Example (2): rDigitValue2

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

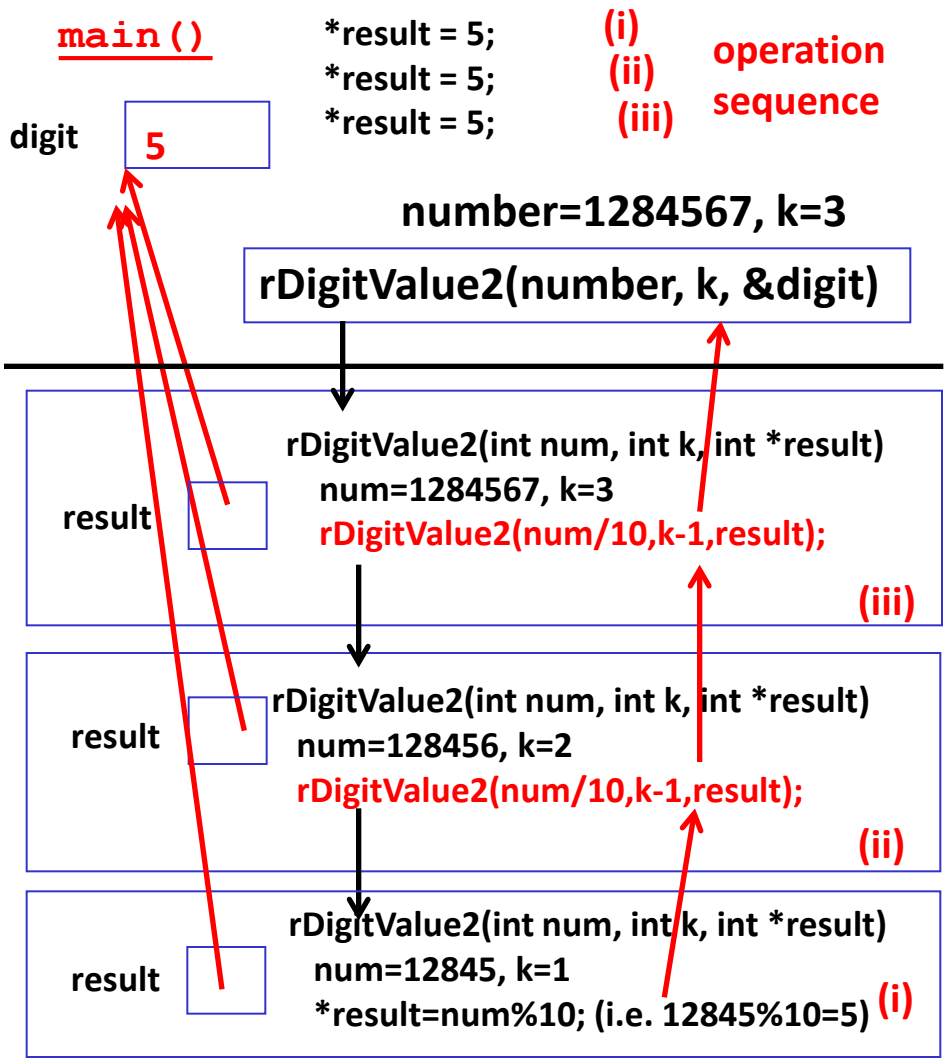


Application Example (2): rDigitValue2

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



Recursion – Q3

```
#include  <stdio.h>
#define BLANK ' '
void saveChar();
int main()
{
   printf("Enter your word and end it
with a space => ");
   saveChar();
   putchar('\n');
   return 0;
}
void saveChar()
{
   char ch;
   ch = getchar();
   if (ch != BLANK)
      saveChar();
   else
      putchar('\n');
   putchar(ch);
}
```

Enter your word and end it with a space => **ward**

draw

Please note that there is a blank character at the end of the input word before the “enter” key is pressed.

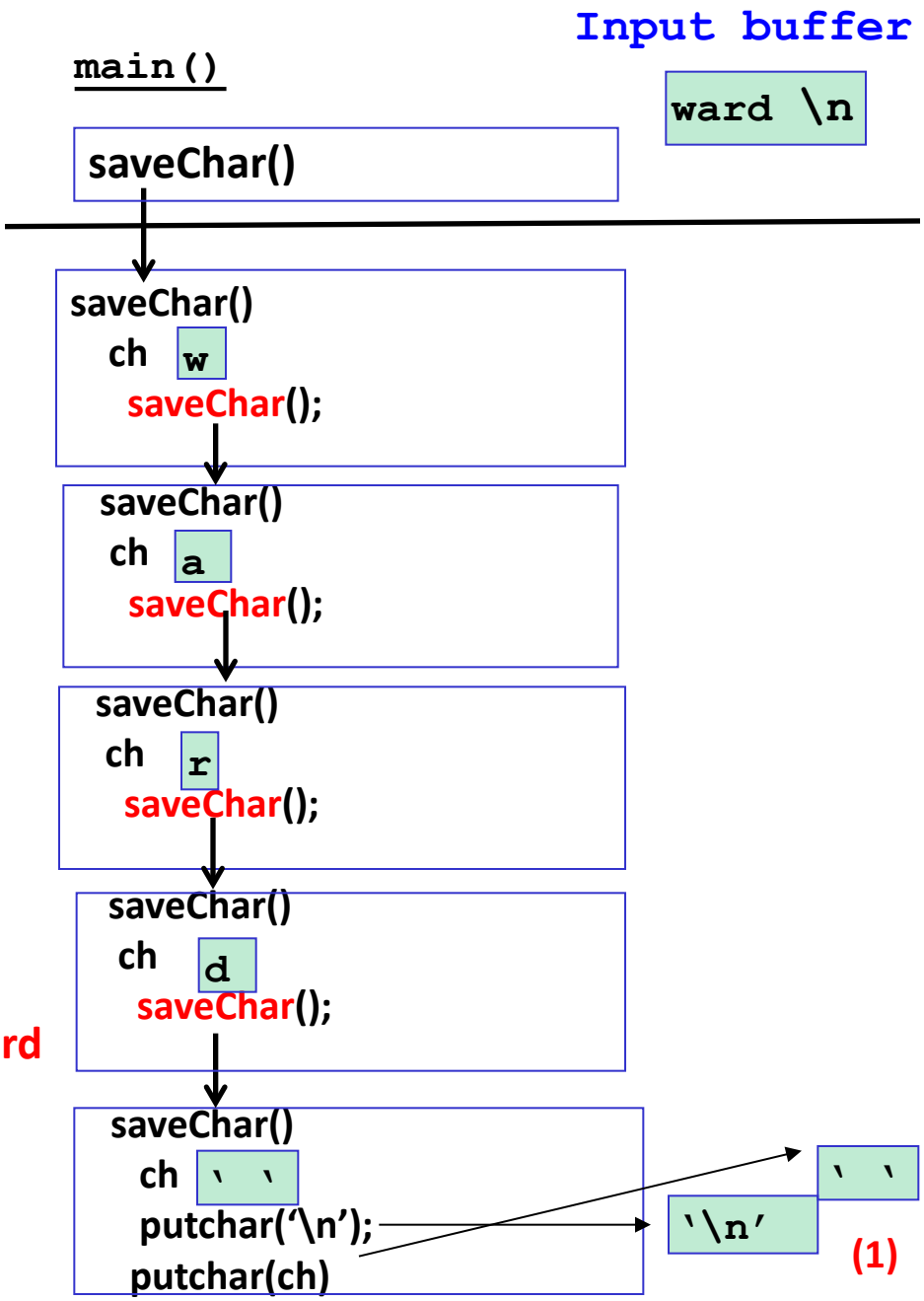
Basically, this program prints an input string, which ends with a space character, in the reversed order.

Q3

```
#include  <stdio.h>
#define BLANK ' '
void saveChar();
int main()
{
   printf("Enter your word and end it
with a space => ");
   saveChar();
   putchar('\n');
   return 0;
}
void saveChar()
{
   char ch;
   ch = getchar();
   if (ch != BLANK)
      saveChar();
   else
      putchar('\n');
   putchar(ch);
}
```

Enter your word and end it with a space => **ward**

draw

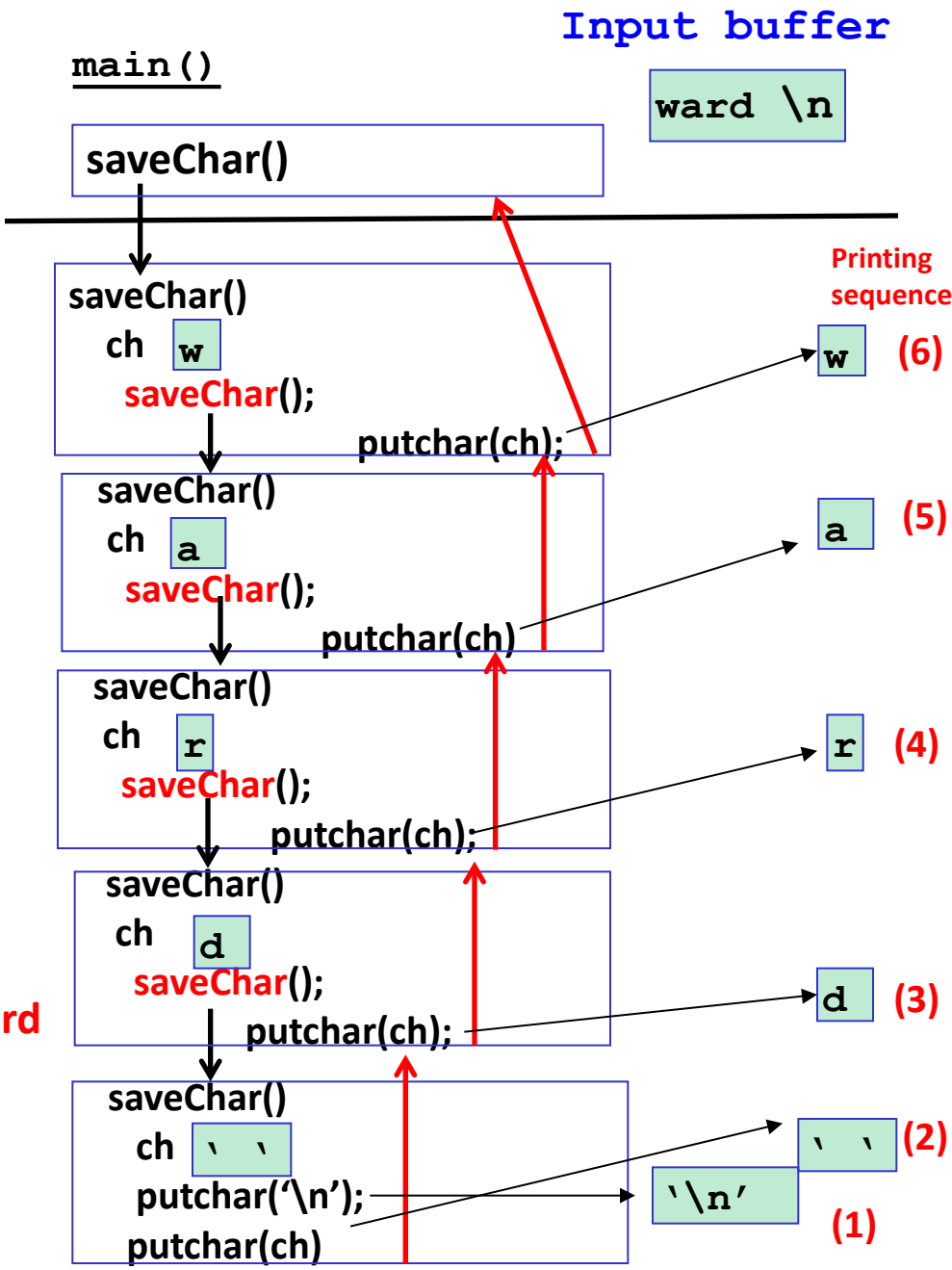


Q3

```
#include  <stdio.h>
#define BLANK ' '
void saveChar();
int main()
{
   printf("Enter your word and end it
with a space => ");
   saveChar();
   putchar('\n');
   return 0;
}
void saveChar()
{
   char ch;
   ch = getchar();
   if (ch != BLANK)
      saveChar();
   else
      putchar('\n');
   putchar(ch);
}
```

Enter your word and end it with a space => **ward**

draw



Q4 (rCountArray)

(**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:

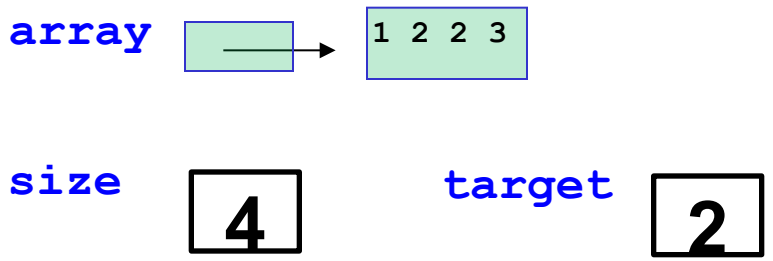
int rCountArray(int array[ ], int *n*, int *a*)

Write a C program to test the functions.

Sample input and output sessions:

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

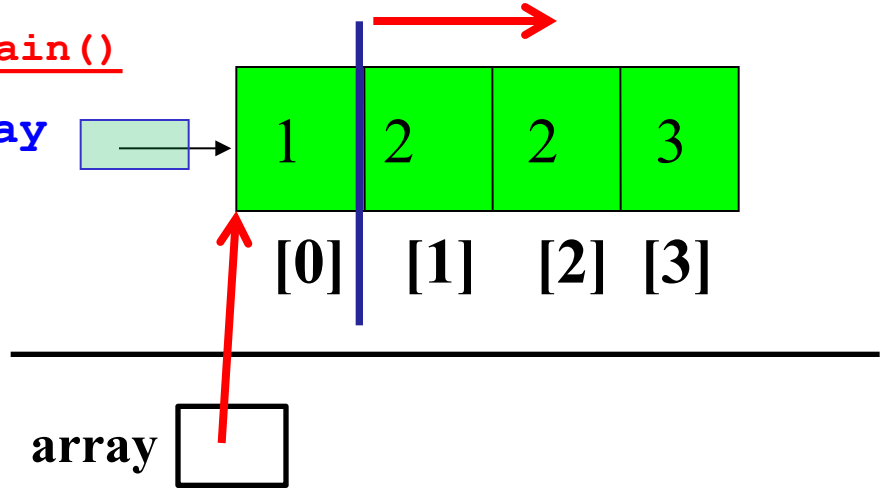
```
#include <stdio.h>
#define SIZE 10
int rCountArray(int array[], int n, int a);
int rCountArray2(int array[], int n, int a);
int main()
{
   int array[SIZE];
   int index, count, target, size;

   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;
}
```



Q4 (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);
}
```



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

Q4 (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)

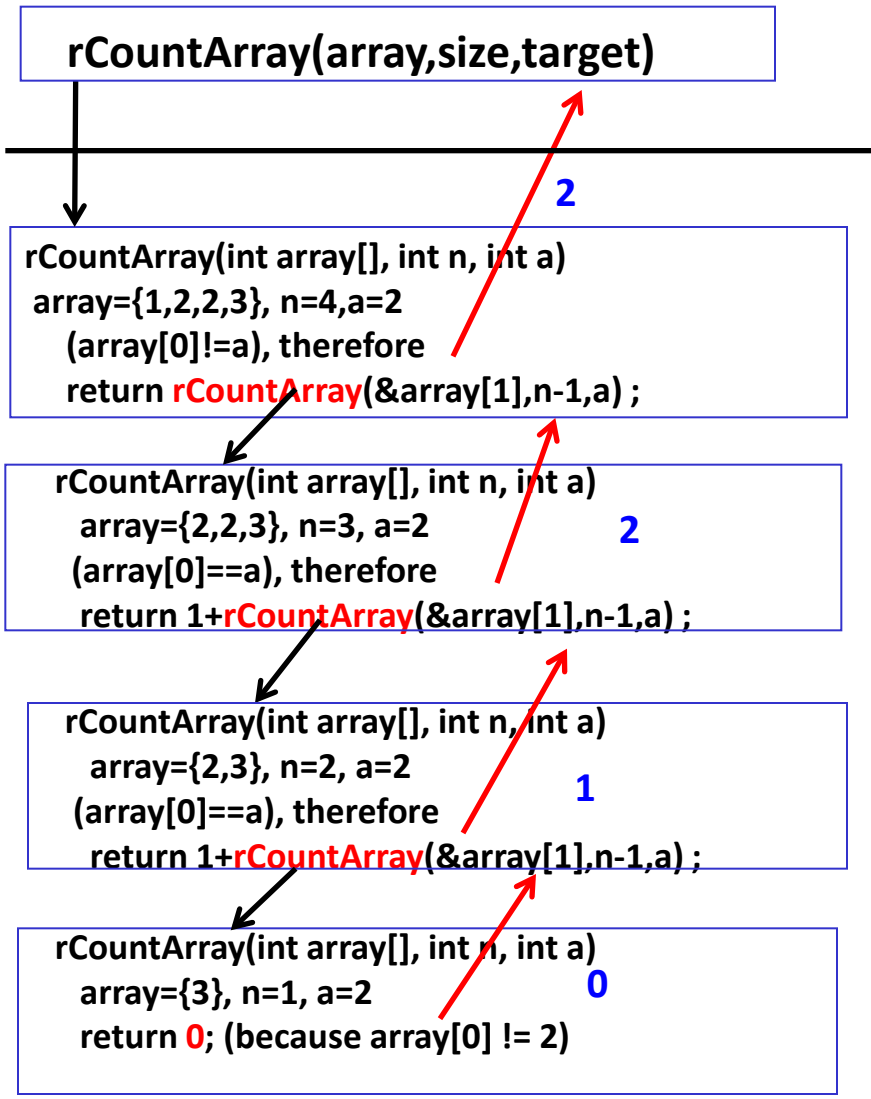
Q4 (rCountArray)

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

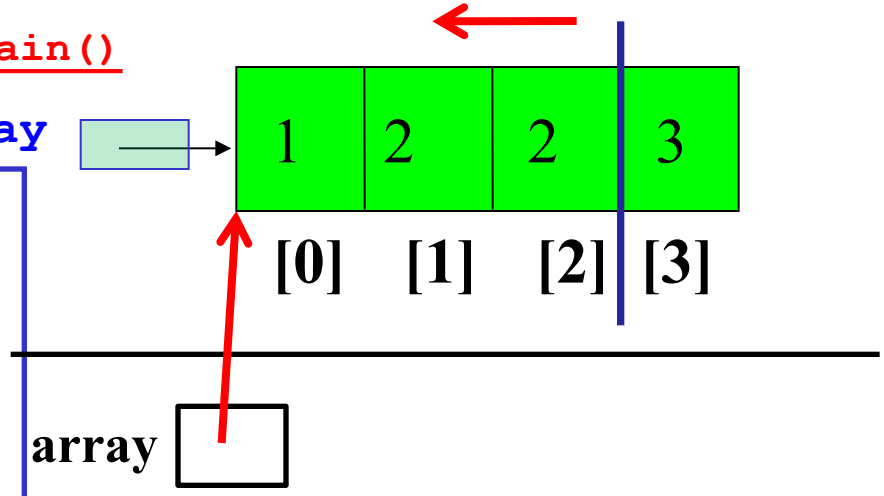
main() array={1,2,2,3}, size=4, target=2



Q4 (rCountArray)

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

Q4 (rCountArray)

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

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)

Q4 (rCountArray)

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