Recursion – 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.

Recursion – Q1 (rSumUp)

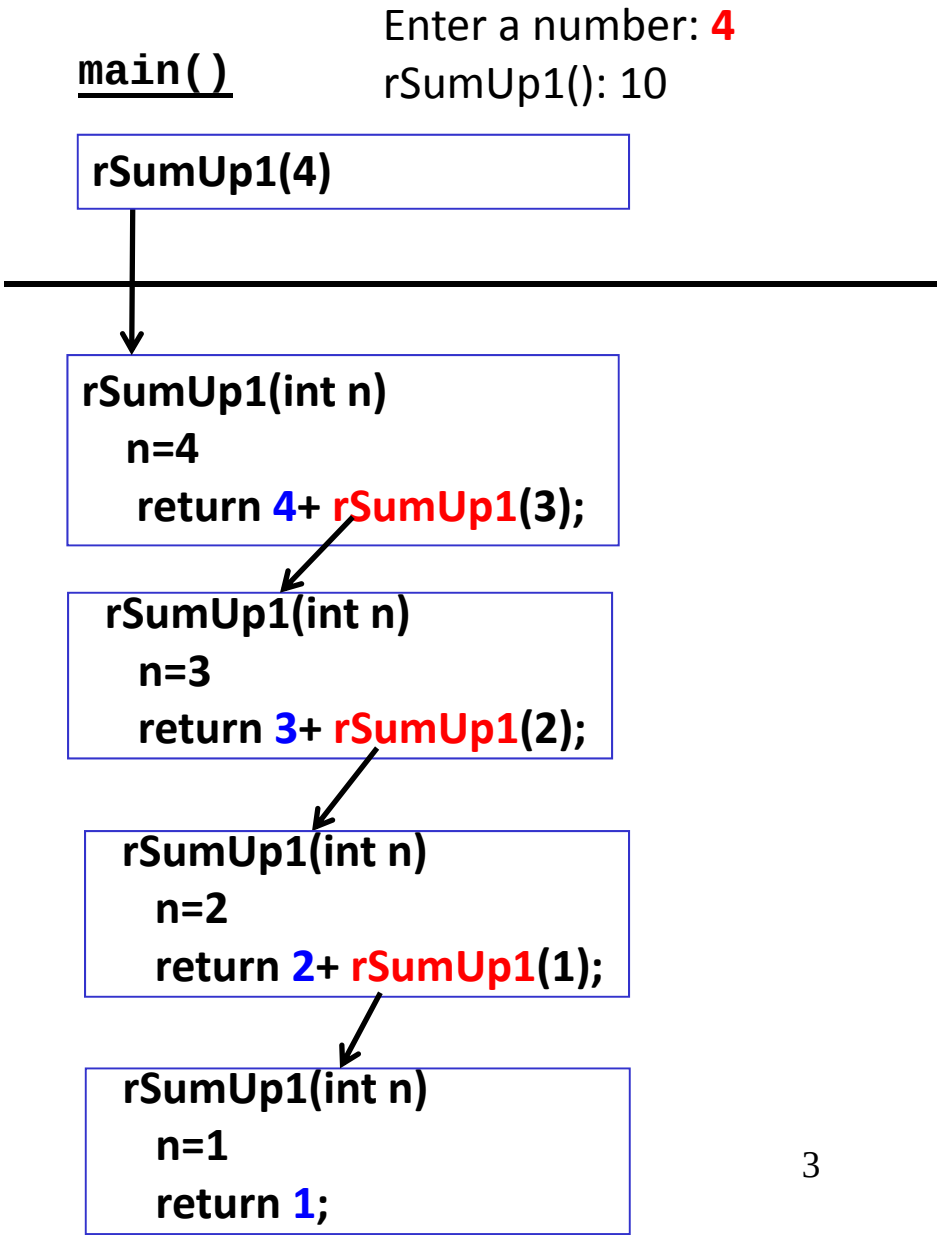
```
#include <stdio.h>
int rSumUp1(int n);
void rSumUp2(int n, int *result);
int main()
{
   int n, result;

   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;
}
```

Recursion – 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

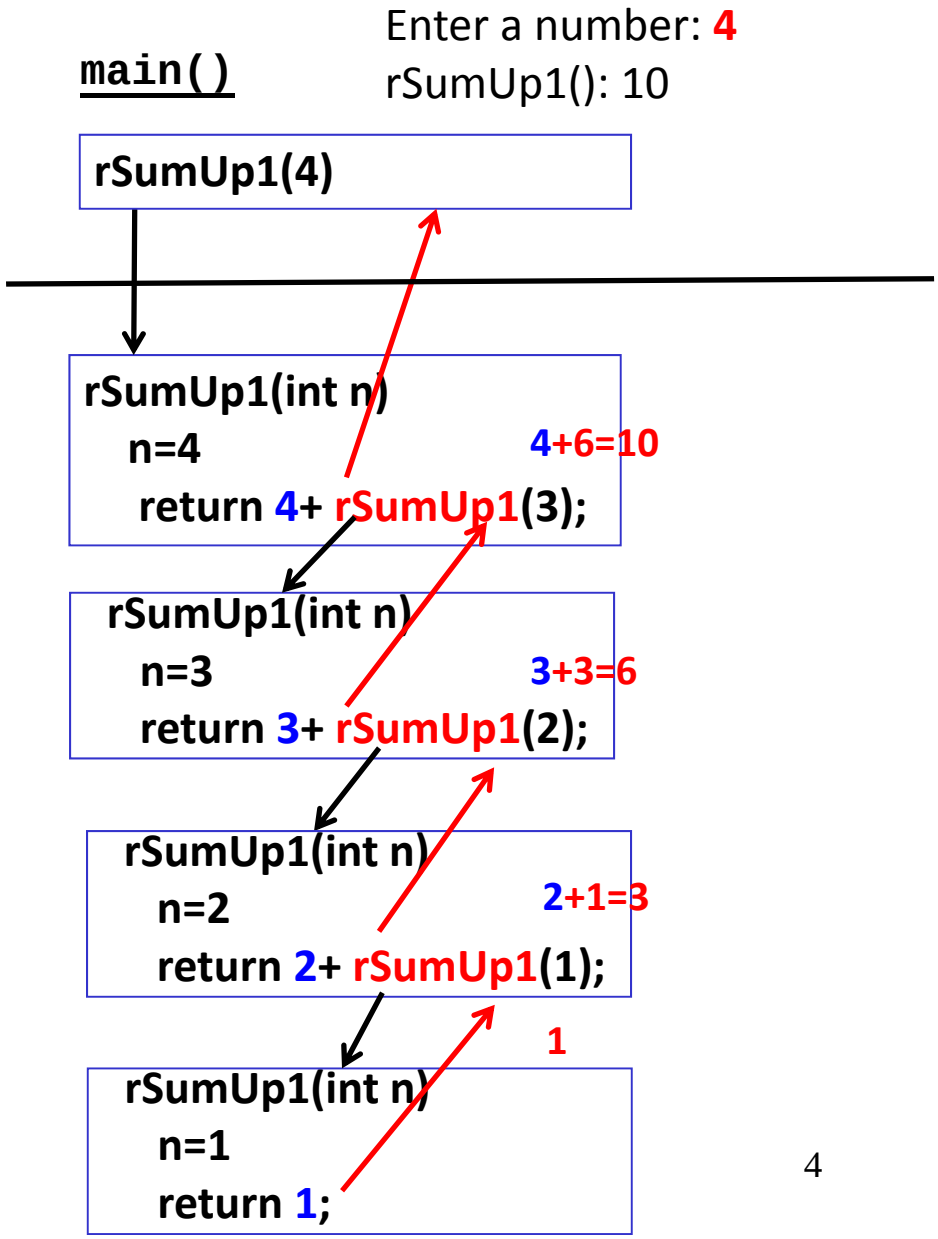


Recursion – Q1 (rSumUp1)

By Returning Value

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

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

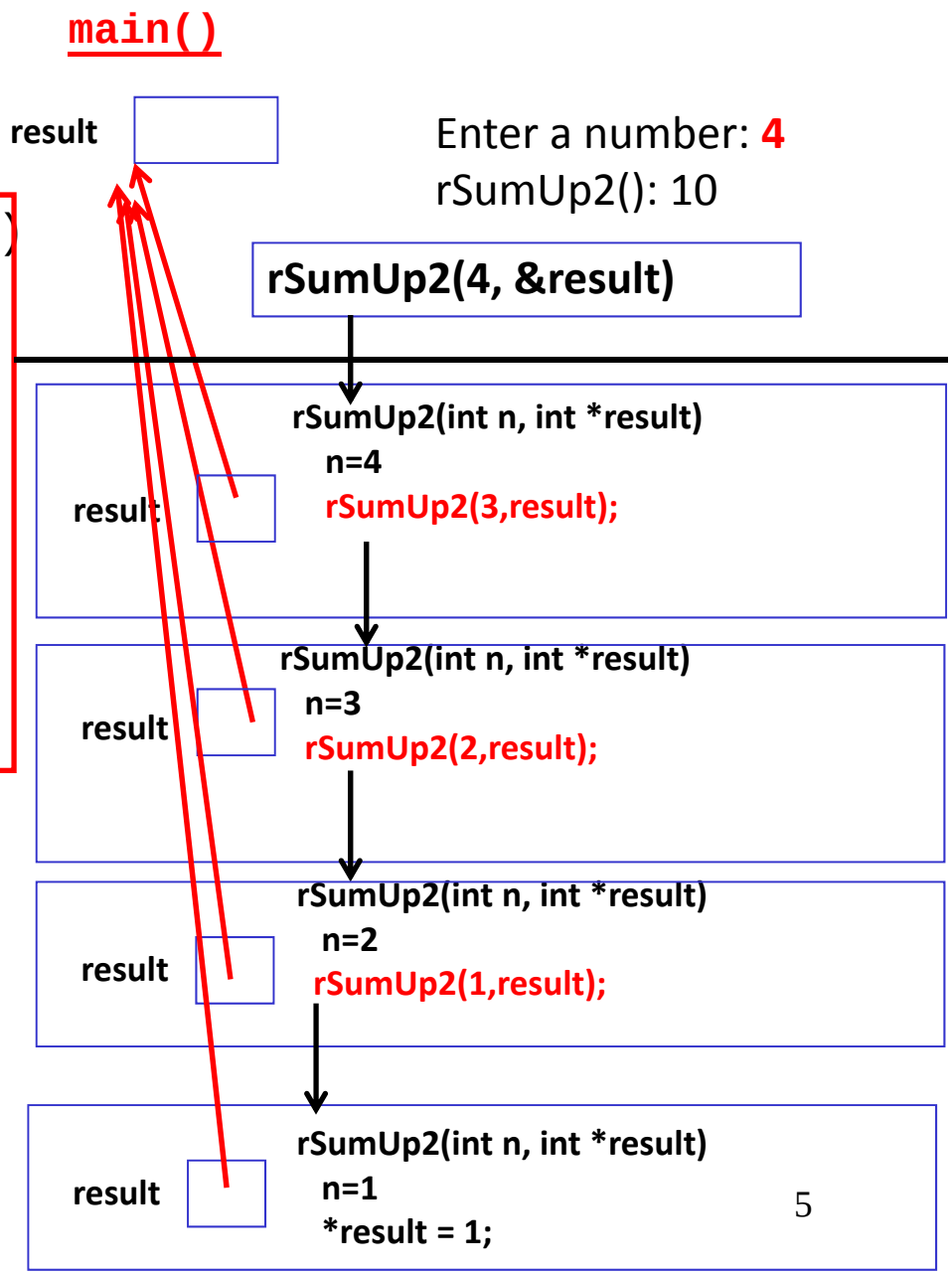


Recursion – Q1 (rSumUp2)

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



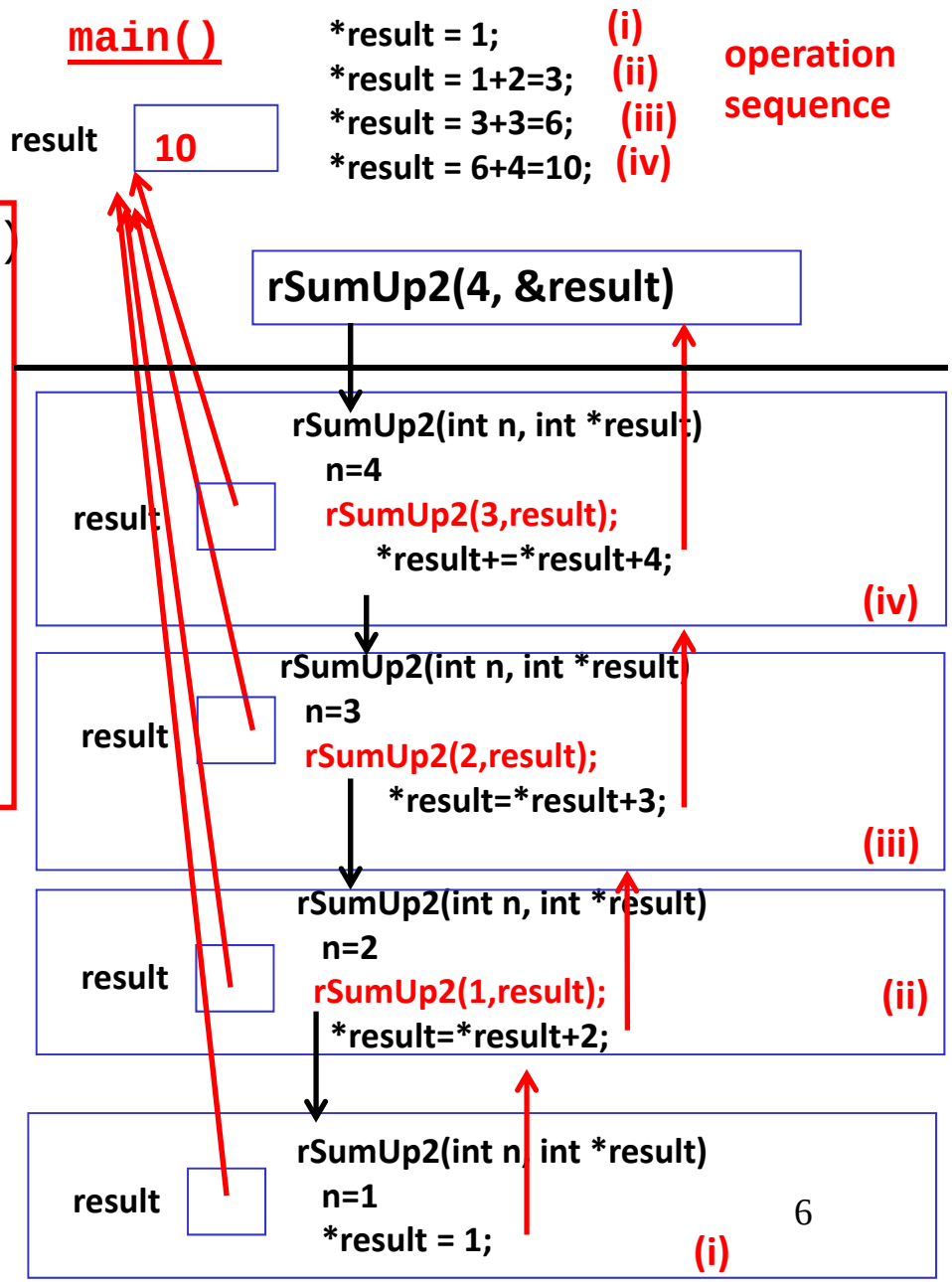
Recursion – Q1 (rSumUp2)

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



Recursion: Q2 (rdigitValue)

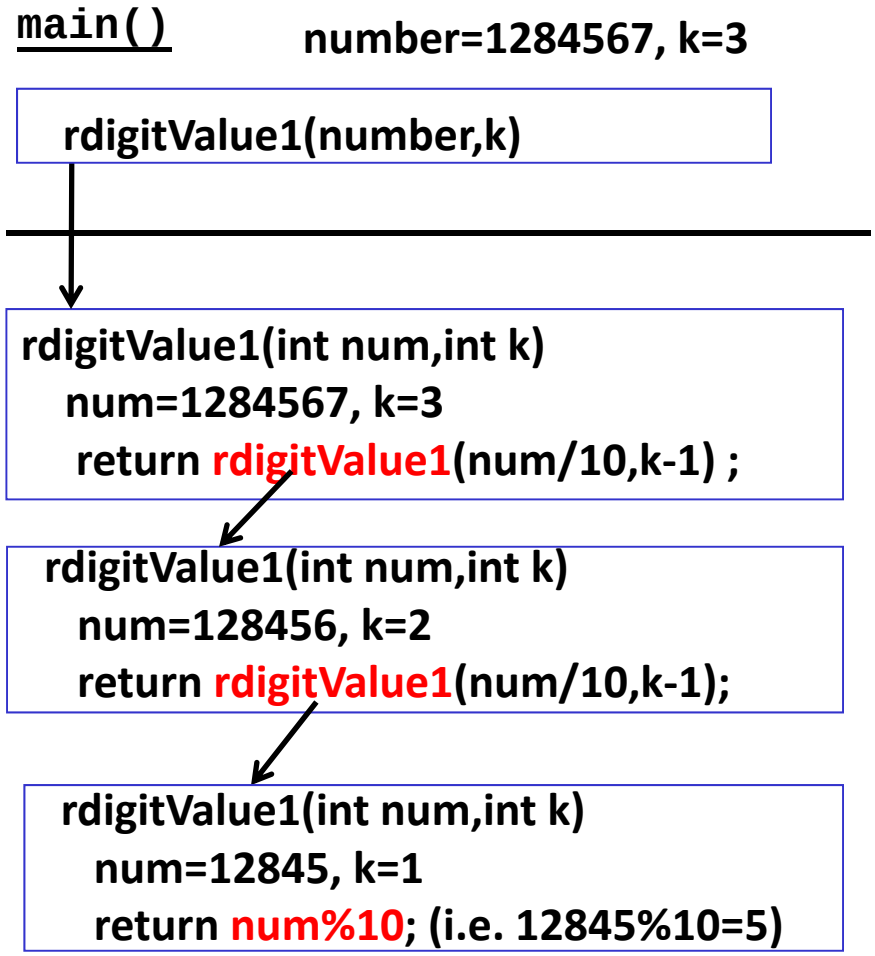
```
#include  <stdio.h>
int rdigitValue1(int num, int k);
void rdigitValue2(int num, int k, int *result);
int main(){
   int k;
   int number, digit;

   printf("Enter a number: ");
   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

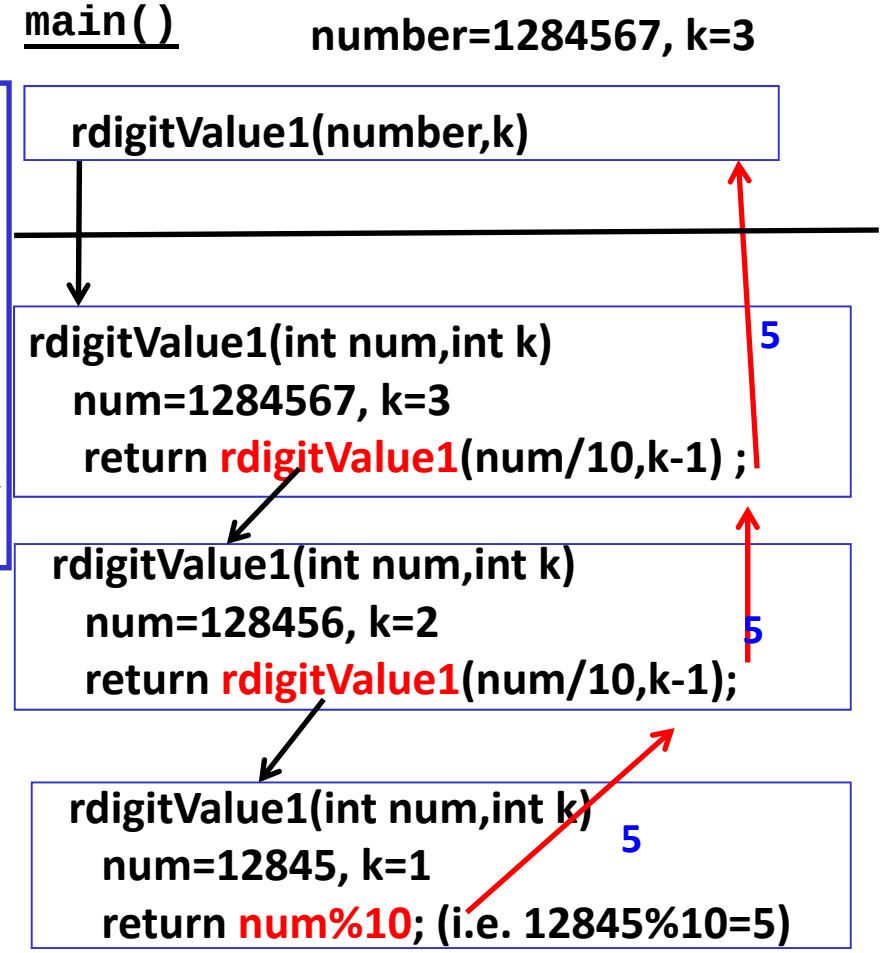


Application Example (2): rdigitValue1

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

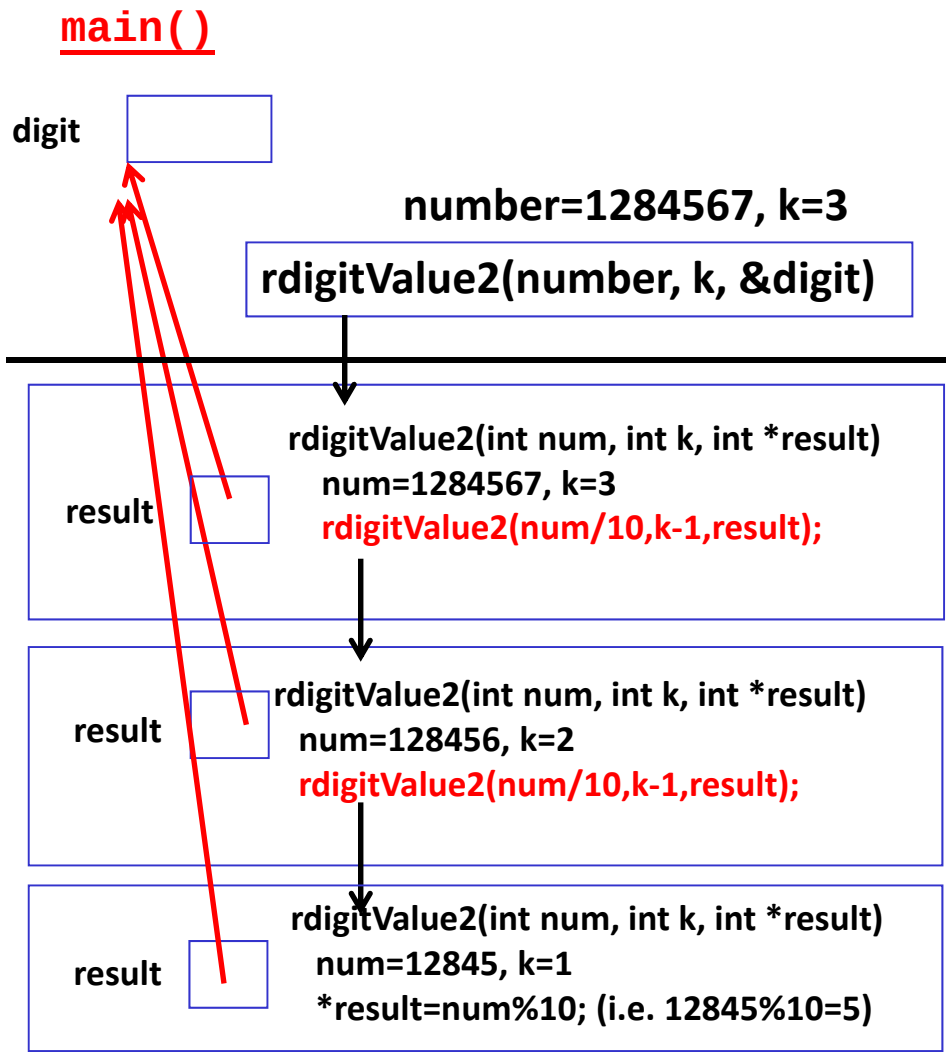


Application Example (2): rdigitValue2

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

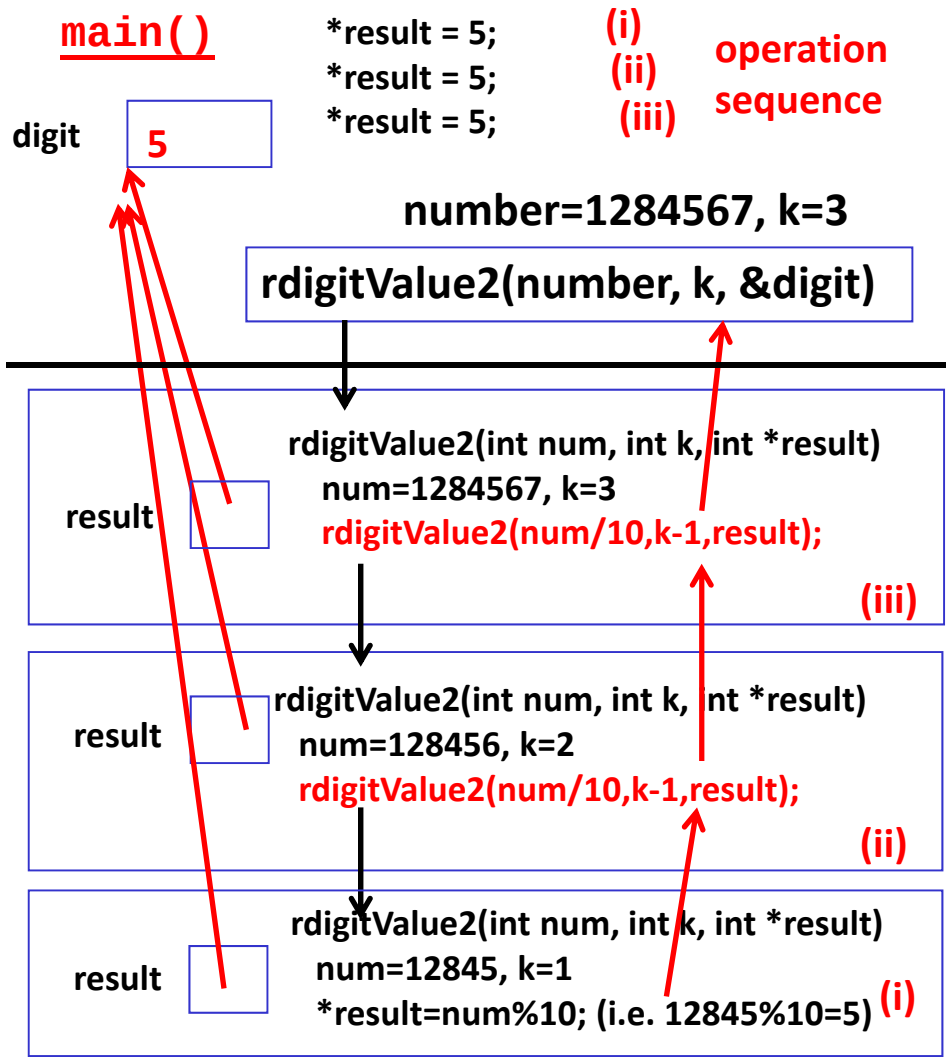


Application Example (2): rdigitValue2

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.

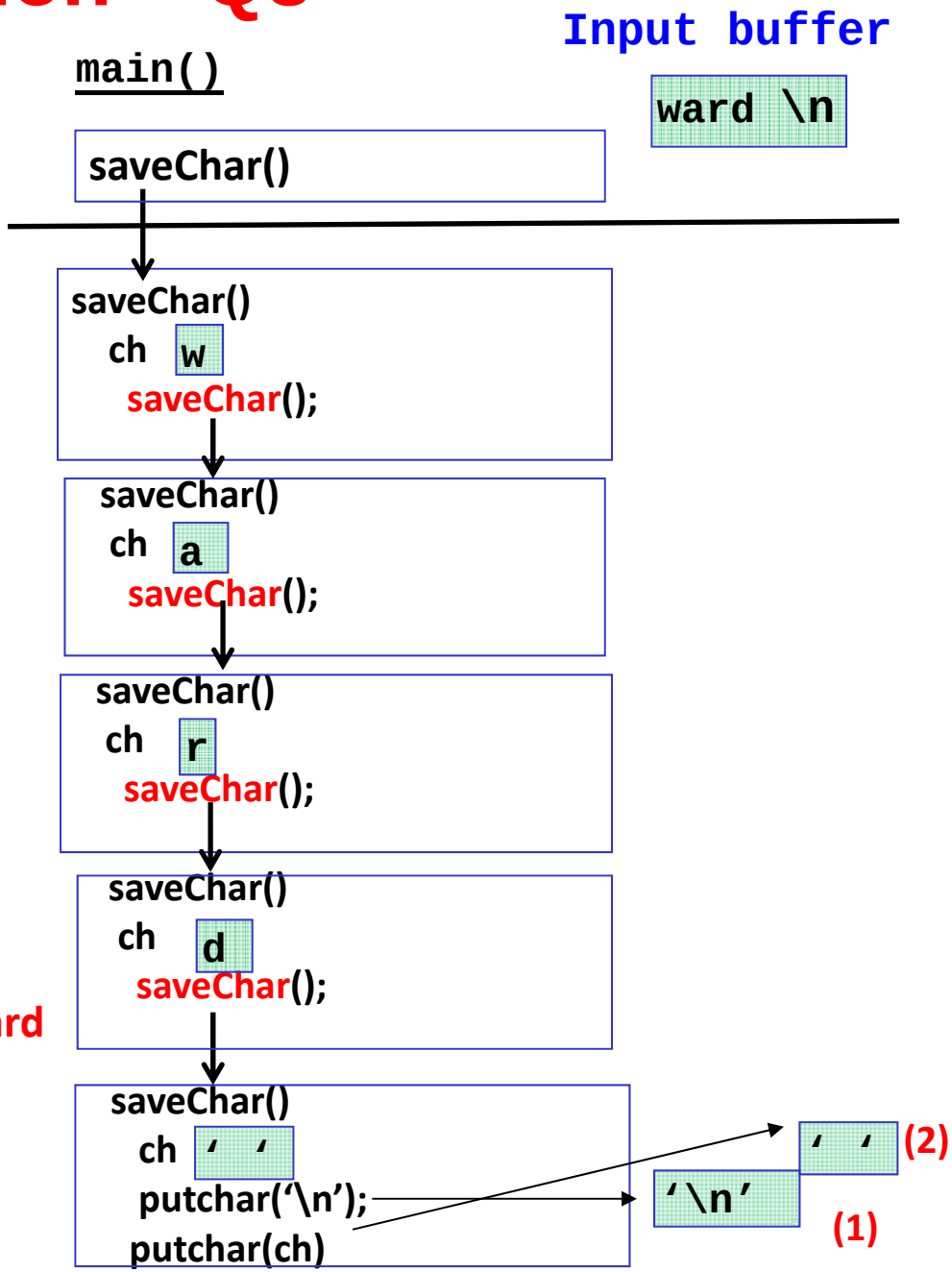
Basically, this program prints an input string, which ends with a space character, in the reversed order.

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

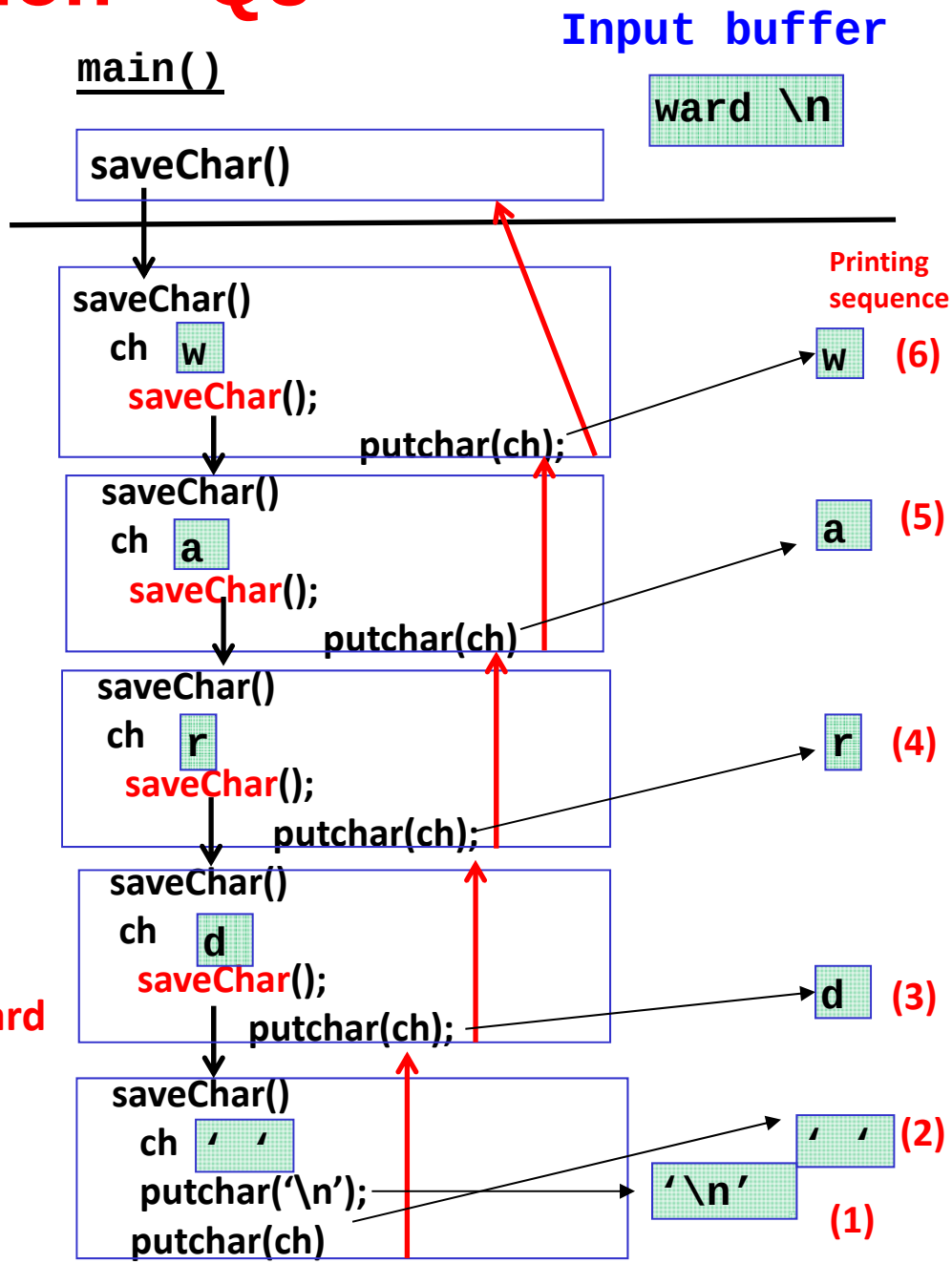


Recursion – Q3

```
#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



Recursion – Q4

(**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:

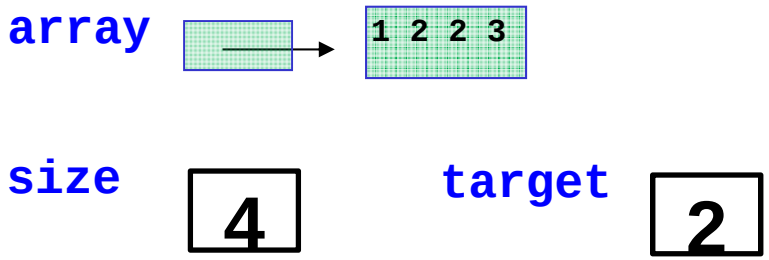
int rCountArray(int array[ ], int *n*, int *a*)

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

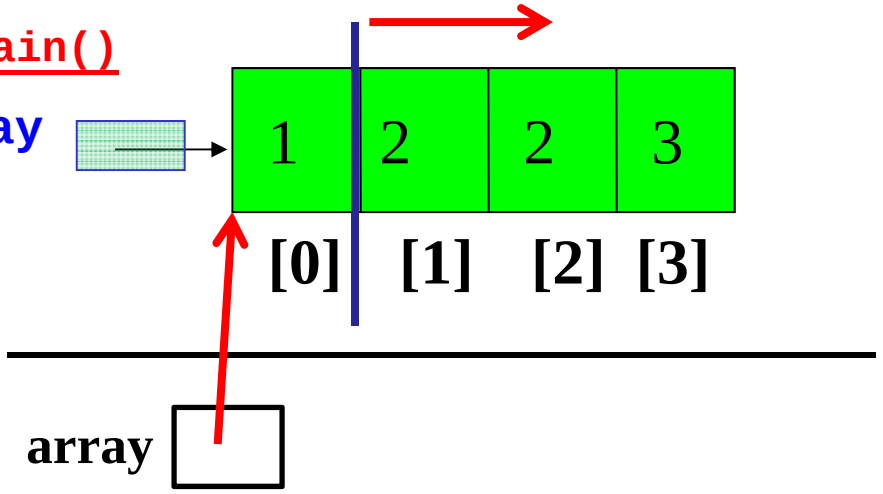
```
#include <stdio.h>
#define SIZE 10
int rCountArray(int array[], int n, int a);
int rCountArray2(int array[], int n, int a);
int main()
{
   int array[SIZE];
   int index, count, target, size;

   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;
}
```



Recursion – Q4

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

Recursion – Q4

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) ;

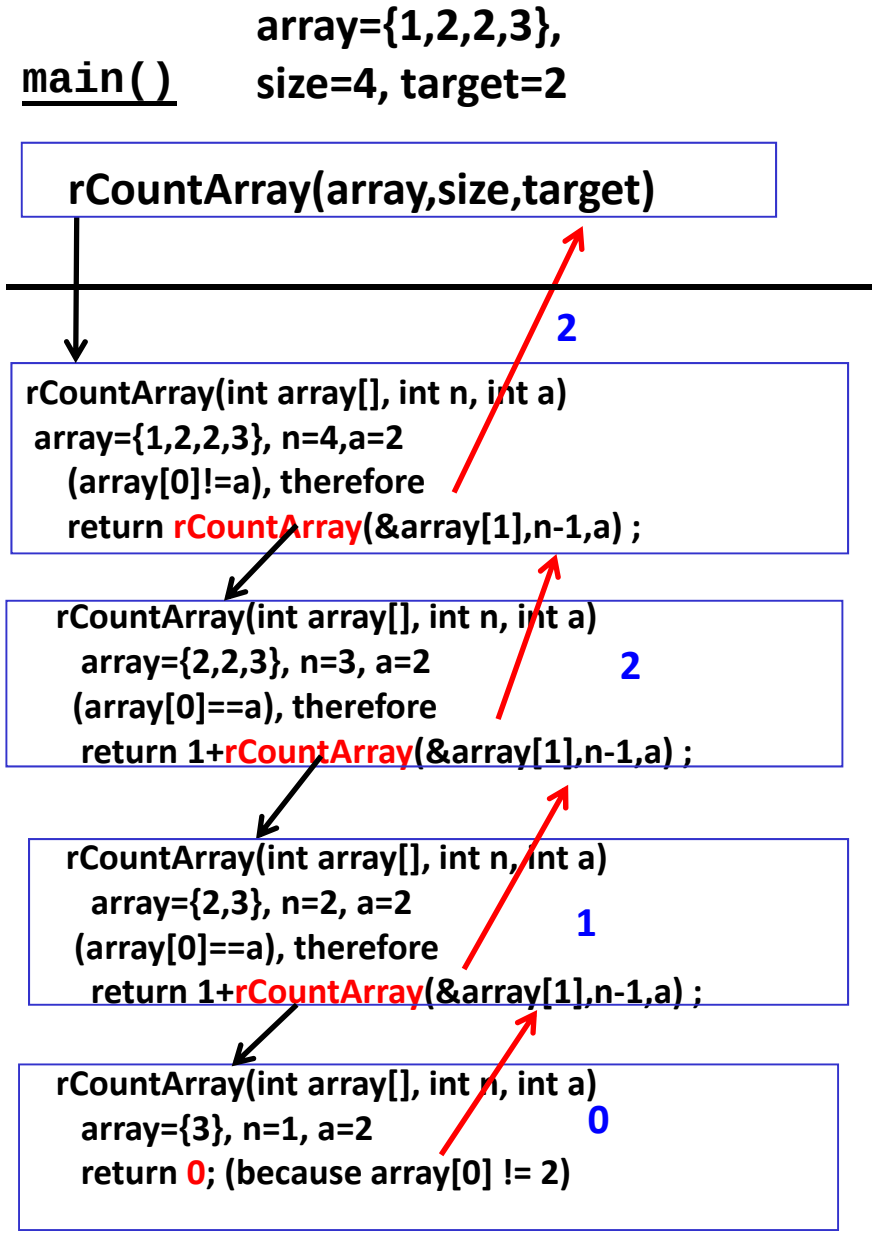
rCountArray(int array[], int n, int a)
array={3}, n=1, a=2
return 0; (because array[0] != 2)

Recursion – Q4

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);
}
```

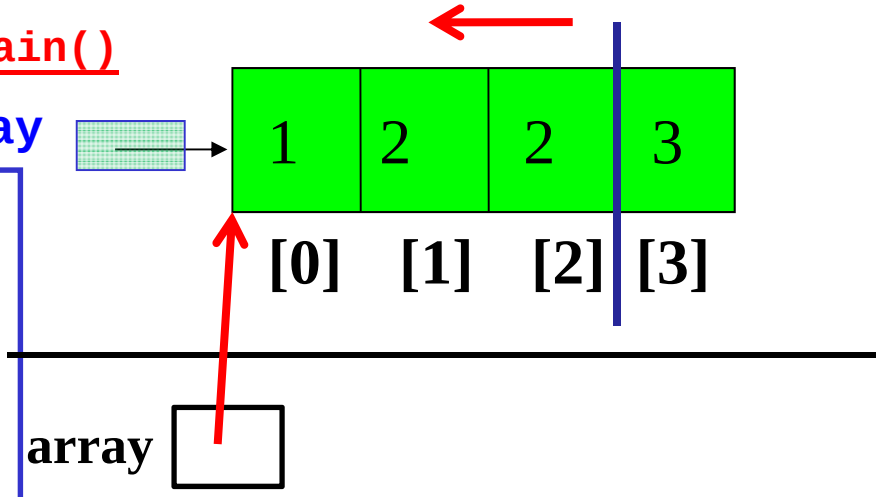
Enter array size: ***4***
Enter 4 numbers: ***1 2 2 3***
Enter the target number: ***2***
rCountArray() = 2



Recursion – Q4

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

Recursion – Q4

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)

Recursion – Q4

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