CSE102 HW04

Part 1 50pts

In this part, you will build an array with some modifications with the given array as input parameter and return it on output array parameter. We want you to write a function (**addPadding**) that takes an array and adds padding (additional array elements) to left and right side of this array. Padding operation is explained below. addPadding function also takes array size (integer), padding width (integer, number of elements that you will add both sides) and padding method (function pointer) as argument. You must implement three different padding methods. These methods are described below:

Example:

inputArr: [5, 6, 7, 8, 9]

-Zero padding: All padding values are 0.

ouputArr: [0, 0, 0, 5, 6, 7, 8, 9, 0, 0, 0], paddingWidth: 3

-Same padding: Padding values are the value of the first element of input array (inputArr[0]) for left padding and the value of the last element of input array (inputArr[arraySize-1]) for right padding.

ouputArr: [5, 5, 5, 5, 5, 6, 7, 8, 9, 9, 9, 9, 9], paddingWidth: 4

-Half padding: Padding values are half of the value of the first element of input array (inputArr[0]/2) for left padding and half of the value of the last element of input array (inputArr[arraySize-1]/2) for right padding.

ouputArr: [2.5, 2.5, 5, 6, 7, 8, 9, 4.5, 4.5], paddingWidth: 2

addPadding function will only copy input array to output array and shift output array by paddingWidth. Padding values are assigned by calling the given function in paddingMethod parameter.

Note: You can assume that output array can have at most 255 elements.

Signature

```
void addPadding(double inputArr[], int inputArraySize, double outputArr[], int
*outputArraySize, int paddingWidth, void paddingMethod(double[], int, int));

void zeroPadding(double outputArr[], int outputArraySize, int paddingWidth);

void samePadding(double outputArr[], int outputArraySize, int paddingWidth);

void halfPadding(double outputArr[], int outputArraySize, int paddingWidth);
```

Sample Usage

```
double inputArr = {5, 6, 7, 8, 9};

double outputArr[255];

int outputArrSize = 0;


addPadding(inputArr, 5 , outputArr, &outputArrSize, 4, samePadding);
```

Return Value

```
ouputArr: [5, 5, 5, 5, 5, 6, 7, 8, 9, 9, 9, 9, 9]
```

Part 2 50pts

In the second part, you will make a convolution operation over a given array. This operation has two components: First component is an input array and second component is a kernel array. The relationship between these two components are shown in figure 1. Kernel array moves on the input array in each iteration and calculates one array element for the output array.

$$one\ element = K_1 * A_i + K_2 * A_{i+1} + \cdots + K_j * A_{i+j-1}$$

If you look at the output array in the figure 1, you can see that the size of the array shrinks after convolution operation. To prevent that, you will add some padding to the inputArray and get an output array with the same size. You can use the function that you wrote in the part 1 for the padding operation.
Another parameter, stride that effects the step for the convolution operation. Stride is the amount of kernel movement for each iteration. Stride is 1 for the figure 1. If stride is 2, then you will move the kernel not one element, but 2 elements for one iteration. If there is no error return 0, otherwise return -1.

PaddingType parameter is an enum type. Definition of that type is given below.

Error condition:

-kernelSize > inputArrSize return -1;

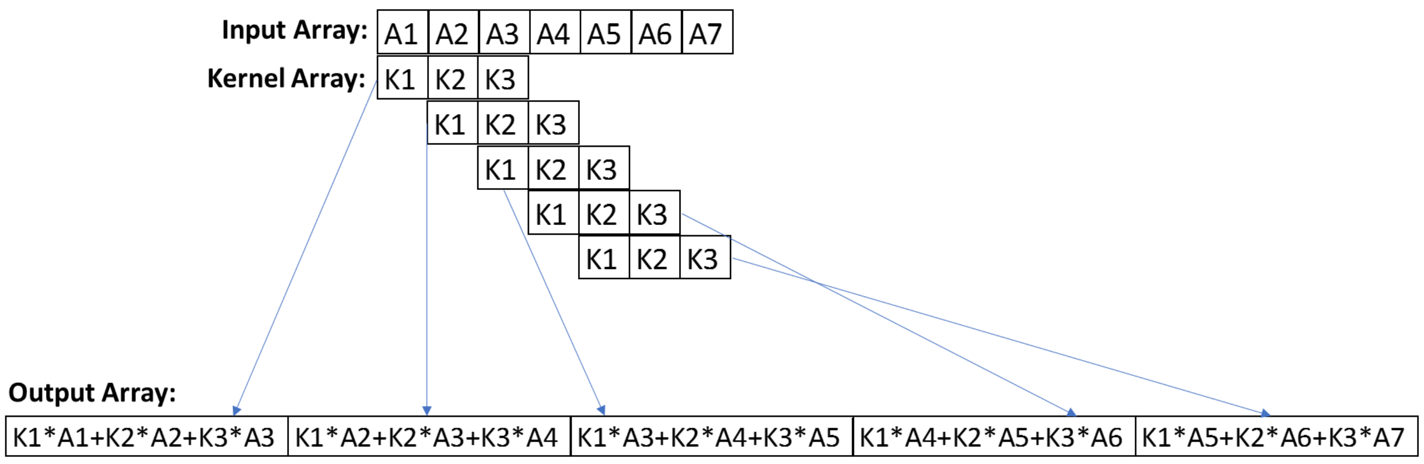

Figure 1: Convolution operation

Signature

```
typedef enum PaddingType = {ZERO, SAME, HALF}
int convolution(double inputArr[], int inputArraySize, double kernelArr[], int
kernelArraySize, double outputArr[], int *outputArraySize, int stride, PaddingType
padding);
```

Sample Usage

```
double inputArr = {3, 5, 7, 9, 11, 13, 15};
double kernelArr = {-1, 2, 0.5, 4, 1.7};
double outputArr[255];
int outputArrSize = 0;

convolution(inputArr, 7, kernelArr, 5, outputArr, &outputArrSize, 1, SAME);
```

Return Value

```
ouputArr: [36.4, 48.8, 65.2, 79.6, 94, 105, 108]
```

Good luck!