Додаток 1: Прва верзија од функцијата за филтрирање

```
/**
 * This is a public-domain C implementation done by Blagoj Kupev.
 *
 * InputData_p - Pointer to input data buffer containing original picture. The same buffer will be
 *               used for transformed picture.
 * Height      - Picture height in pixels
 * Width       - Picture width in pixels
 * wSize       - Median window size in pixels
 */
Error_t MedianFilter (u8 * const InputData_p, const u32 Height, const u32 Width, const u32 wSize)
{
  u32 PosX;
  u32 PosY;
  u32 HalfRadius;
  u8 * Window_p = NULL;
  char WinX, WinY;
  register u32 Temp;
  u32 Temp1;

  if ((Radius & 0x01) == 0)
  {
    DEBUG_0(printf ("{main.c}[%d]: Even radius entered!\n", __LINE__););
    return EVEN_RADIUS;
  }
  if ((Radius > Width) || (Radius > Height))
  {
    DEBUG_0(printf("{main.c}[%d]: Invalid dimensions entered!\n", __LINE__););
    return INVALID_DIMENSIONS;
  }

  Window_p = malloc(Radius*Radius);
  if (Window_p == NULL)
  {
    DEBUG_0(printf ("{main.c}[%d]: Circular buffer allocation error!\n", __LINE__););
    return MALLOC_ERROR;
  }

  HalfRadius = Radius/2;
  if (InitCircBuff((HalfRadius*Width) + HalfRadius))
  {
    DEBUG_0(printf ("{main.c}[%d]: Circular buffer allocation error!\n", __LINE__););
    return CIRC_BUFF_ERROR;
  }

  for (PosY = 0; PosY<Height; PosY++)
  {
    for (PosX = 0; PosX<Width; PosX++)
    {
      u32 WinIndex = 0;
      Temp = Width - HalfRadius + 1;
      if ((PosX==(1)) && (PosY == 1))
      {
        DEBUG_0(printf("Test point reached\n"););
      }

      WinY = Radius;
      while (WinY != 0)
      {
        WinY--;
        if (((PosY + WinY) < (HalfRadius)) ||
            ((PosY + WinY) > Temp))
        {
          continue;
        }
        WinX = Radius;
        while (WinX != 0 )
        {
          WinX--;
          if (((PosX + WinX) < (HalfRadius)) ||
              ((PosX + WinX) > Temp))
          {
            continue;
          }

          if (WinY > HalfRadius)
          {
```

```
            if (((PosX + WinX - HalfRadius) < Width) || ((PosY + WinY - HalfRadius) < Height))
            {
              Window_p[WinIndex]          =          InputData_p[(PosX+WinX-HalfRadius)+((PosY+WinY-
HalfRadius)*Width)];
            }
            else
            {
              continue;
            }
          }
          else
          {
            if (WinY < HalfRadius)
            {
              Window_p[WinIndex]=ReadCircBuffEnd((HalfRadius    -    WinX    -    1)+((HalfRadius    -
WinY)*Width));
            }
            else if ((WinY == HalfRadius) && (WinX < HalfRadius))
            {
              Window_p[WinIndex]   =   ReadCircBuffEnd((HalfRadius   -   WinX   -   1)+((HalfRadius-
WinY)*Width));
            }
            else if (((PosX + WinX - HalfRadius) < Width) || ((PosY + WinY - HalfRadius) < Height))
            {
              Window_p[WinIndex]          =          InputData_p[(PosX+WinX-HalfRadius)+((PosY+WinY-
HalfRadius)*Width)];
            }
            else
            {
              continue;
            }
          }
          WinIndex++;
        }
      }

      Temp1 = PosX+(PosY*Width);
      AddToCircBuffer(InputData_p[Temp1]);
      InputData_p[Temp1] = CalcMedian(Window_p, WinIndex);
    }
  }
  ReleaseCircBuff();
  return SUCCESS;
}

static u32 CalcMedian(u8 * const Window_p, const u32 NrOfElements)
{
  (void)quickSort(Window_p, NrOfElements);
  return Window_p[NrOfElements>>1];
}

//  quickSort
//
//  This public-domain C implementation by Darel Rex Finley.
//
//  * Returns YES if sort was successful, or NO if the nested
//    pivots went too deep, in which case your array will have
//    been re-ordered, but probably not sorted correctly.
//
//  * This function assumes it is called with valid parameters.
//
//  * Example calls:
//    quickSort(&myArray[0],5); // sorts elements 0, 1, 2, 3, and 4
//    quickSort(&myArray[3],5); // sorts elements 3, 4, 5, 6, and 7

static Error_t quickSort(u8 *arr, u8 elements)
{

  #define  MAX_LEVELS  1000

  int  piv, beg[MAX_LEVELS], end[MAX_LEVELS], i=0, L, R ;

  beg[0]=0; end[0]=elements;
  while (i>=0) {
    L=beg[i]; R=end[i]-1;
    if (L<R) {
      piv=arr[L]; if (i==MAX_LEVELS-1) return UNSUCESSFULL_SORTING;
      while (L<R) {
        while (arr[R]>=piv && L<R) R--; if (L<R) arr[L++]=arr[R];
        while (arr[L]<=piv && L<R) L++; if (L<R) arr[R--]=arr[L]; }
```

```
      arr[L]=piv; beg[i+1]=L+1; end[i+1]=end[i]; end[i++]=L; }
    else {
      i--; }}
  return SUCCESS;
}
```