Увод в програмирането

11: Основни операции с масиви

доц. Атанас Семерджиев

Съдържание

- Left Shift, Right Shift
- Добавяне на елемент
- Премахване на елемент
- Сортиране
- Намиране на елемент

2

Добавяне и премахване на елемент

3

Добавяне на елемент

4

```
void ShiftRight(double* pArray,
                size_t Size,
                size_t StartFrom,
                size_t Positions)
{
    size_t write = Size + Positions - 1;
    size_t read  = Size - 1;

    while (read >= StartFrom)
    {
        pArray[write--] = pArray[read--];
    }
}
```

```
void InsertAt(double *pArr,
              size_t Size,
              size_t Index,
              double Element)
{
    ShiftRight(pArr, Size, Index, 1);

    pArr[Index] = Element;
}
```

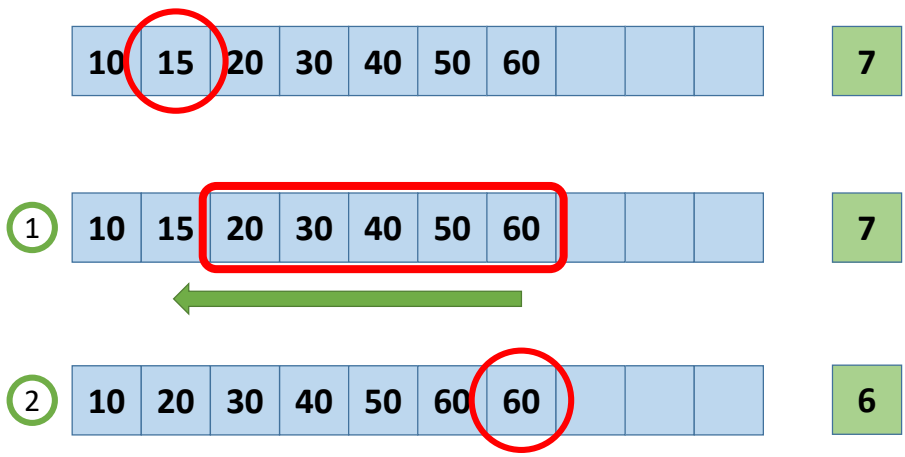
```
void PrintArray(const double * pArr, size_t Size)
{
    for (size_t i = 0; i < Size; i++)
    {
        std::cout << pArr[i] << std::endl;
    }
}

void main()
{
    double data[10] = { 10, 20, 30, 40, 50, 60 };

    InsertAt(data, 6, 1, 15);

    PrintArray(data, 10);
}
```

Премахване на елемент



8

```
void ShiftLeft(double* pArr,
               size_t Size,
               size_t StartFrom,
               size_t Positions)
{
    size_t write = StartFrom;
    size_t read  = StartFrom + Positions;

    while (read < Size)
    {
        pArr[write++] = pArr[read++];
    }
}
```

```
void RemoveAt(double* pArr, size_t Size, size_t Index)
{
    ShiftLeft(pArr, Size, Index, 1);
}

void main()
{
    double data[10] = { 10, 20, 30, 40, 50, 60 };
    RemoveAt(data, 6, 0);
    PrintArray(data, 10);
}
```

Сортиране

11

```
// Пряка селекция

void SelectionSort(double* pArr, size_t Size)
{
    size_t Min;

    for (size_t i = 0; i < Size; i++)
    {
        Min = i;

        for (size_t j = i + 1; j < Size; j++)
        {
            if (pArr[j] < pArr[Min])
                Min = j;
        }

        if (Min != i)
            std::swap(pArr[i], pArr[Min]);
    }
}
```

```
// Метод на мехурчето

void BubbleSort(double* pArr, size_t Size)
{
    size_t Rightmost = Size - 1;

    for (size_t i = 0; i < Rightmost; i++)
    {
        for (size_t j = Rightmost; j > i; j--)
        {
            if (pArr[j - 1] > pArr[j])
                std::swap(pArr[j - 1], pArr[j]);
        }
    }
}
```

Търсене на елемент

14

Просто търсене

```
int Find(const double* pArr, size_t Size, double Element)
{
    for (size_t i = 0; i < Size; i++)
    {
        if (pArr[i] == Element)
            return i;
    }

    return -1;
}
```

Двоично търсене

- Масивът трябва да е сортиран
- Връща се първият намерен елемент

16

```
int BinarySearch(const double* pArr, size_t Size, double Element)
{
    size_t Left = 0, Right = Size;

    while (Left <= Right) {
        int Middle = (Left + Right) / 2;

        if (pArr[Middle] == Element)
            return Middle;
        else if (Element > pArr[Middle])
            Left = Middle + 1;
        else
            Right = Middle - 1;
    }
    return -1;
}
```