

# Увод в програмирането

Масиви. Основни операции с масиви  
ФМИ, специалност „Софтуерно инженерство“

Слайдовете в тази презентация са създадени от  
доц. Атанас Семерджиев

# Съдържание

- Тип Масив. Дефиниция
- Основни операции с масиви
- Многомерни масиви
- Работа с паметта. Динамична памет

# Типове от данни

- По-рано през семестъра въведохме типове данни за числа и символи. Елементите на тези структури се състоят от една компонента и се наричат **прости**, или **скаларни**.
- Структури от данни, компонентите на които са редици от елементи, се наричат **съставни**.
- Структури от данни, за които операциите добавяне и изтриване на елемент не са допустими, се наричат **статични**, в противен случай - **динамични**.

# Структура от данни масив ...

- **Физическо представяне**

Елементите на масива се записват последователно в паметта на компютъра, като за всеки елемент на редицата се отделя определено количество памет.

В езика C++ структурата масив се реализира чрез типа масив

# Тип масив ...

- *Примери:*

- `Int a[5]`

- дефинира масив от 5 елемента от тип `int`, индексирани от 0 до 4;

- `Double d[10]`

- дефинира масив от 10 елемента от тип `double`, индексирани от 0 до 9;

- `Bool b[4]`

- дефинира масив от 4 елемента от тип `bool`, индексирани от 0 до 3.

```
int x = 1;
```

```
int array[10]; // int[10]
```

```
array[0] = 100; // Първата клетка
```

```
array[1] = 100; // Втората клетка
```

```
array[9] = 100; // Последната клетка
```

```
// Името на масива е указател към първия му елемент
```

```
*array = 100; // Първата клетка, екв. на *(array+0)
```

```
*(array+1) = 100; // Втората клетка
```

```
*(array+9) = 100; // Последната клетка
```

```
void PrintArray(double arr[5])
{
    for (int i = 0; i < 5; i++)
    {
        std::cout << arr[i] << std::endl;
    }
}

void main()
{
    double data[5] = { 1., 2., 3., 4., 5. };

    PrintArray(data);
}
```

```
void PrintArray(double arr[5])  
{  
    for (int i = 0; i < 5; i++)  
    {  
        std::cout << arr[i] << std::endl;  
    }  
}
```

```
void main()  
{  
    double data[3] = { 1., 2., 3. };  
    PrintArray(data);  
}
```



```
void PrintArray(double arr[], int Size)
{
    for (int i = 0; i < Size; i++)
    {
        std::cout << arr[i] << std::endl;
    }
}

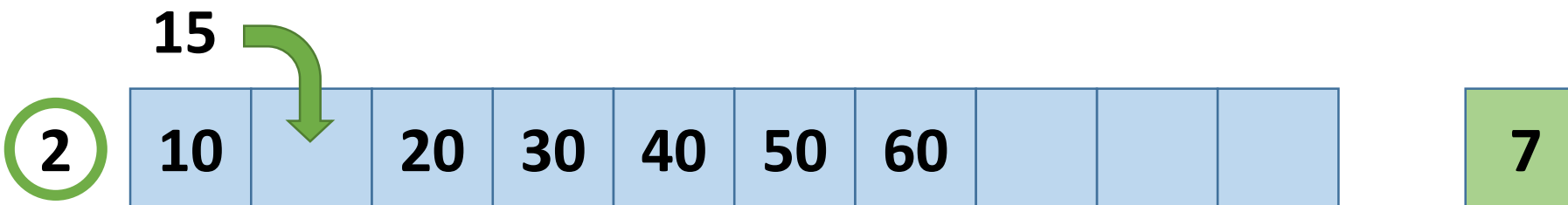
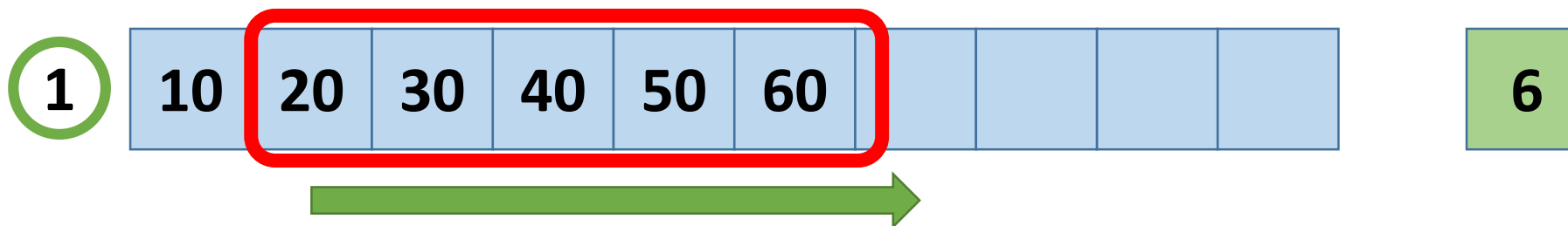
void main()
{
    double dataA[3] = { 1., 2., 3. };
    double dataB[5] = { 1., 2., 3., 4., 5. };

    PrintArray(dataA, 3);
    PrintArray(dataB, 5);
}
```

# Основни операции с масиви

- Добавяне на елемент
- Премахване на елемент
- Търсене на елемент
- Сортиране

# Добавяне на елемент



```
void ShiftRight(double* pArray,  
               size_t Size,  
               size_t StartFrom,  
               size_t Positions)  
{  
    if (pArray == nullptr) return;  
  
    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)  
{  
    if (pArr == nullptr) return;  
  
    ShiftRight(pArr, Size, Index, 1);  
  
    pArr[Index] = Element;  
}
```

```
void PrintArray(const double * pArr, size_t Size)
{
    if (pArr == nullptr) return;

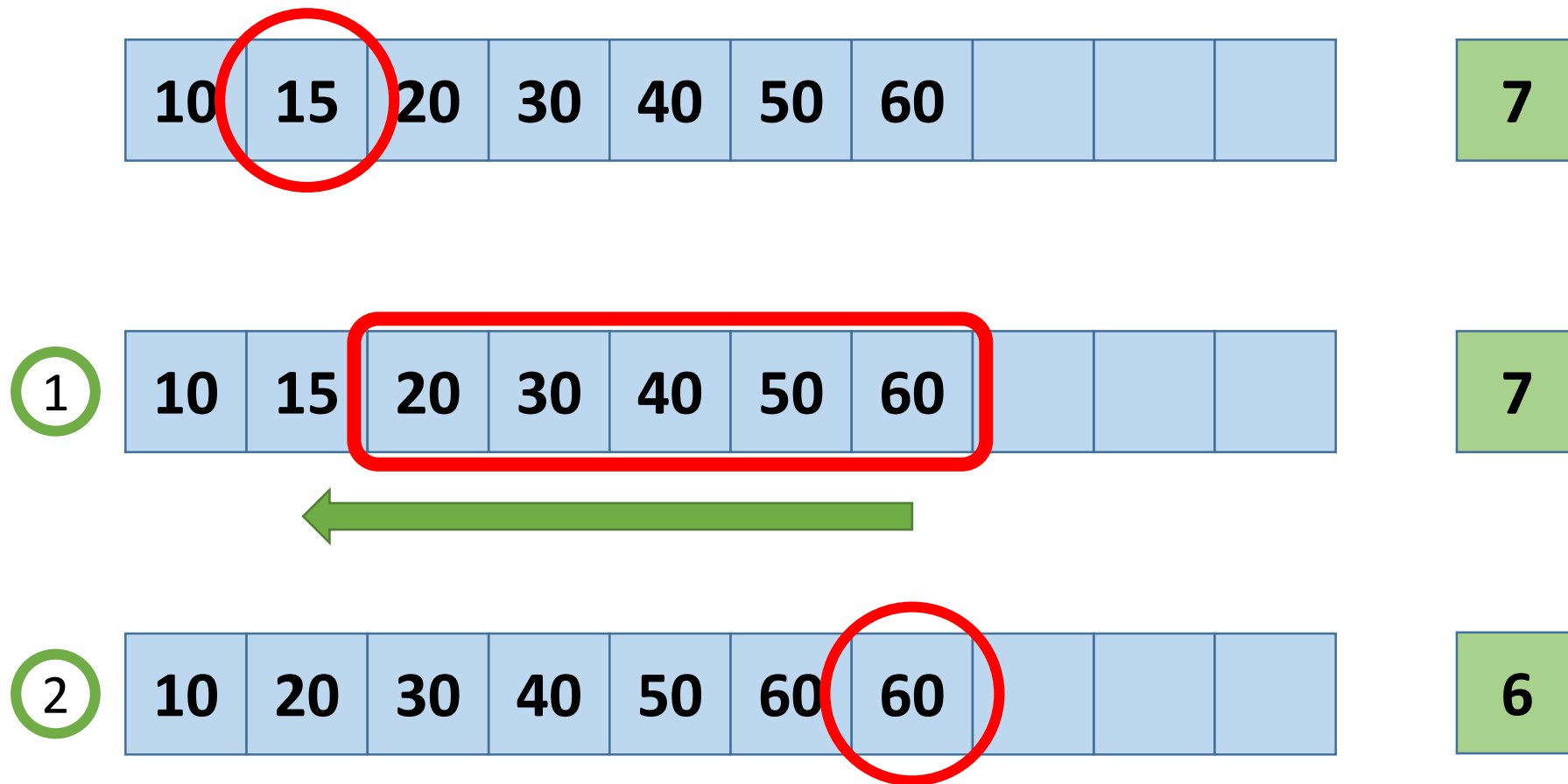
    for (size_t i = 0; i < Size; i++)
    {
        std::cout << pArr[i] << std::endl;
    }
}

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

    InsertAt(data, 6, 1, 15);

    PrintArray(data, 10);
}
```

# Премахване на елемент



```
void ShiftLeft(double* pArr,  
               size_t Size,  
               size_t StartFrom,  
               size_t Positions)  
{  
    if (pArr == nullptr) return;  
  
    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)
{
    if (pArr != nullptr)
        ShiftLeft(pArr, Size, Index, 1);
}
```

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

# Търсене на елемент

# Просто търсене

```
int Find(const double* pArr, size_t Size, double Element)
{
    if (pArr == nullptr) return -2;

    for (size_t i = 0; i < Size; i++)
    {
        if (pArr[i] == Element)
            return i;
    }

    return -1; // Елементът не е намерен
}
```

# Двоично търсене

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

```
int BinarySearch(const double* pArr, size_t Size, double Element)
{
    if (pArr == nullptr) return -2;

    size_t Left = 0, Right = Size - 1;

    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; // Елементът не е намерен
}
```

# Сортиране

// Пряка селекция

```
void SelectionSort(double* pArr, size_t Size)
{
    if (pArr == nullptr) return;

    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)
{
    if (pArr == nullptr) return;

    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]);
        }
    }
}
```



# Двумерни масиви



```
double data[3][4] = { 10., 11., 12., 13.,  
                      20., 21., 22., 23.,  
                      30., 31., 32., 33. };
```

```
data[0][2] <--> (*data + 0 * 4 + 2)  
data[1][2] <--> (*data + 1 * 4 + 2)  
data[2][2] <--> (*data + 2 * 4 + 2)  
...
```

```

void PrintArray(double arr[])
{
    for (size_t row = 0; row < 3; row++)
    {
        for (size_t col = 0; col < 5; col++)
        {
            std::cout << arr[row][col] << ' ';
        }
        std::cout << std::endl;
    }
}

```

*\*(arr + row \* ??? + col)*

```

void main()
{
    double data[3][5] = { 10., 11., 12., 13., 14.,
                          20., 21., 22., 23., 24.,
                          30., 31., 32., 33., 34. };

    PrintArray(data);
}

```

```

void PrintArray(double arr[][5])
{
    if (arr == nullptr) return;

    for (size_t row = 0; row < 3; row++)
    {
        for (size_t col = 0; col < 5; col++)
        {
            std::cout << arr[row][col] << ' ';
        }
        std::cout << std::endl;
    }
}

void main()
{
    double data[3][5] = { 10., 11., 12., 13., 14.,
                          20., 21., 22., 23., 24.,
                          30., 31., 32., 33., 34. };

    PrintArray(data);
}

```

*\*(arr + row \* 5 + col)*

# Достъп чрез указател

```
void ZeroArray(double * pArray, size_t Size)
{
    if (pArray == nullptr) return;

    for (size_t i = 0; i < Size; i++)
        pArray[i] = 0;
}

void main()
{
    double data[500];
    ZeroArray(data, 500);
}
```

# Достъп чрез указател (const)

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

void main()
{
    double data[5] = { 1., 2., 3., 4., 5. };
    PrintArray(data, 5);
}
```

```
void PrintArray(const double * pArray, size_t Size)
{
    if (pArray == nullptr) return;

    for (size_t i = 0; i < Size; i++)
        cout << pArray[i] << endl;
}

void main()
{
    double data[3][4] = { 10., 11., 12., 13.,
                          20., 21., 22., 23.,
                          30., 31., 32., 33. };

    PrintArray((const double*)data, 3 * 4);
}
```

```
void PrintArray(const double * pArray,  
               size_t Rows,  
               size_t Cols)  
{  
    if (pArray == nullptr) return;  
  
    for (size_t row = 0; row < Rows; row++)  
    {  
        for (size_t col = 0; col < Cols; col++)  
        {  
            std::cout << *(pArray + row * Cols + col)  
                        << ' ';  
        }  
    }  
  
    std::cout << std::endl;  
}
```



```
void main()
{
    double data[3][4] = { 10., 11., 12., 13.,
                          20., 21., 22., 23.,
                          30., 31., 32., 33. };

    PrintArray((const double*)data, 3, 4);
}
```

```

void PrintArray(const double * pArray,
               size_t Rows,
               size_t Cols)
{
    if (pArray == nullptr) return;

    for (size_t row = 0; row < Rows; row++)
    {
        for (size_t col = 0; col < Cols; col++)
        {
            std::cout << pArray[row * Cols + col]
                      << ' ';
        }

        std::cout << std::endl;
    }
}

```

```

// *(pArray + row*Cols + col) <--> pArray[row*Cols + col]

```