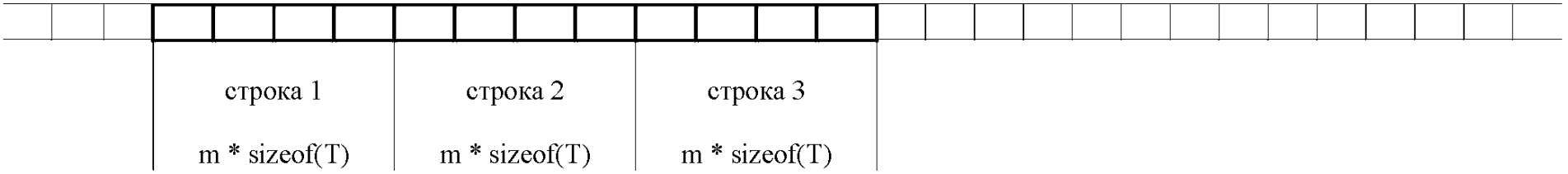
Динамические матрицы

Матрица как одномерный массив

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



k	0	1	2	3	4	5	6	7	8	9	10	11
i;j	0;0	0;1	0;2	0;3	1;0	1;1	1;2	1;3	2;0	2;1	2;2	2;3

a[i][j] ⇔ a[k], k = i * m + j

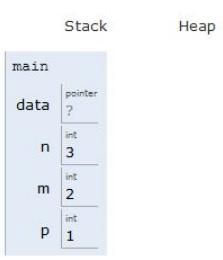
Матрица как одномерный массив

```
double *data;
int n = 2, m = 3;

data = malloc(n * m * sizeof(double));
if (data)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            // Обращение к элементу i, j
            data[i * m + j] = 0.0;

    free(data);
}
```

Матрица как одномерный массив

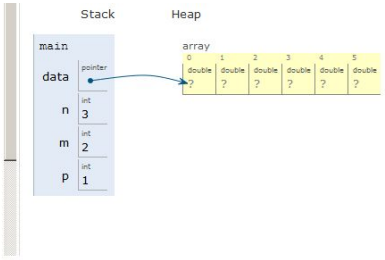
1. Перед выделением памяти

```
4  int main(void)
5  {
6      double *data;
7      int n = 3, m = 2, p = 1;
8
9      // Выделение памяти под "матрицу"
10     data = malloc(n * m * sizeof(double));
11     if (data)
12     {
13         for (int i = 0; i < n; i++)
14         {
15             for (int j = 0; j < m; j++)
16                 // Обращение к элементу i, j
```



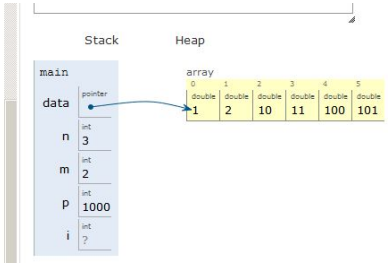
2. Сразу после выделения памяти

```
4  int main(void)
5  {
6      double *data;
7      int n = 3, m = 2, p = 1;
8
9      // Выделение памяти под "матрицу"
10     data = malloc(n * m * sizeof(double));
11     if (data)
12     {
13         for (int i = 0; i < n; i++)
14         {
15             for (int j = 0; j < m; j++)
16                 // Обращение к элементу i, j
```



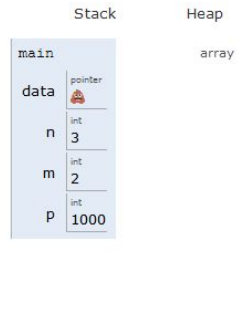
3. Использование выделенной памяти

```
14         {
15             for (int j = 0; j < m; j++)
16                 // Обращение к элементу i, j
17                 data[i * m + j] = p + j;
18
19             p *= 10;
20         }
21
22         for (int i = 0; i < n; i++)
23         {
24             for (int j = 0; j < m; j++)
25                 printf(" %5.1f", data[i * m + j]);
26
```



4. Сразу после освобождения

```
19             p *= 10;
20         }
21
22         for (int i = 0; i < n; i++)
23         {
24             for (int j = 0; j < m; j++)
25                 printf(" %5.1f", data[i * m + j]);
26
27             printf("\n");
28         }
29
30         // Освобождение памяти
31         free(data);
```



Матрица как одномерный массив

- Преимущества:
  - Простота выделения и освобождения памяти.
  - Возможность использовать как одномерный массив.

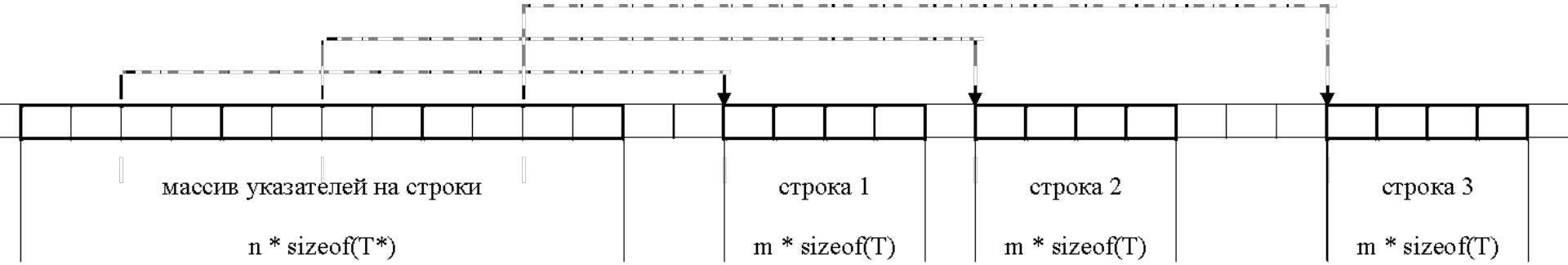
- Недостатки:
  - Средство проверки работы с памятью (СПРП), например Doctor Memory, не может отследить выход за пределы строки.
  - Нужно писать i * m + j, где m – число столбцов.

Матрица как массив указателей

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



Матрица как массив указателей

Алгоритм выделения памяти

Вход: количество строк (n) и количество столбцов (m)

Выход: указатель на массив строк матрицы (p)

- Выделить память под массив указателей (p)
- Обработать ошибку выделения памяти
- В цикле по количеству строк матрицы (0 <= i < n)
  - Выделить память под i-ую строку матрицы (q)
  - Обработать ошибку выделения памяти
  - p[i]=q

Матрица как массив указателей

Алгоритм освобождения памяти

Вход: указатель на массив строк матрицы (p) и количество строк (n)

- В цикле по количеству строк матрицы (0 <= i < n)
  - Освободить память из-под i-ой строки матрицы
- Освободить память из-под массива указателей (p)

Матрица как массив указателей

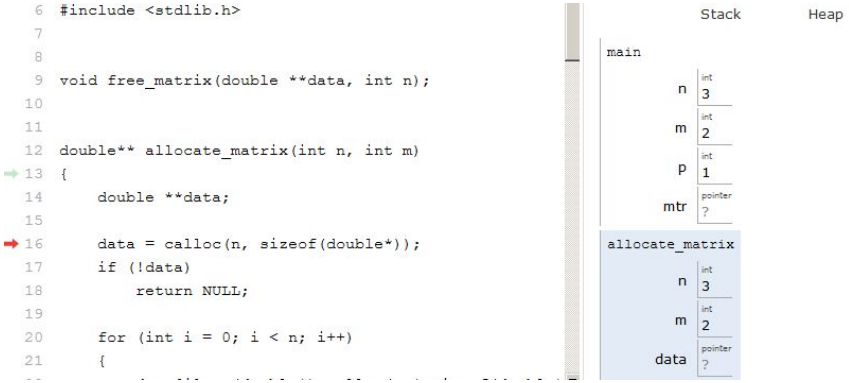
```
void free_matrix(double **data, int n);

double** allocate_matrix(int n, int m)
{
    double **data = calloc(n, sizeof(double*));
    if (!data)
        return NULL;

    for (int i = 0; i < n; i++)
    {
        data[i] = malloc(m * sizeof(double));
```

Матрица как массив указателей

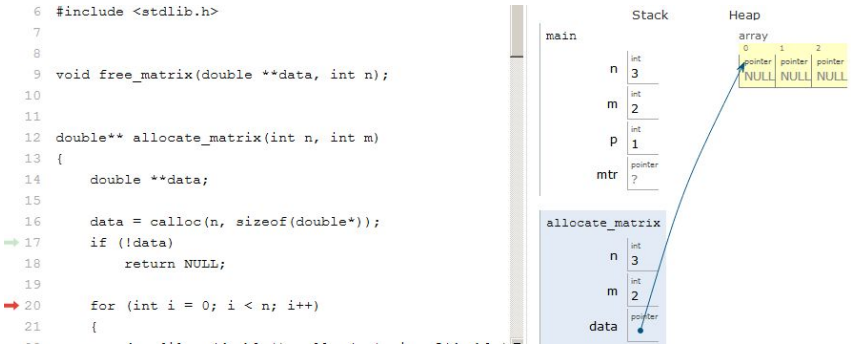
```
        if (!data[i])
        {
            free_matrix(data, n);

            return NULL;
        }
    }

    return data;
}
```

Матрица как массив указателей

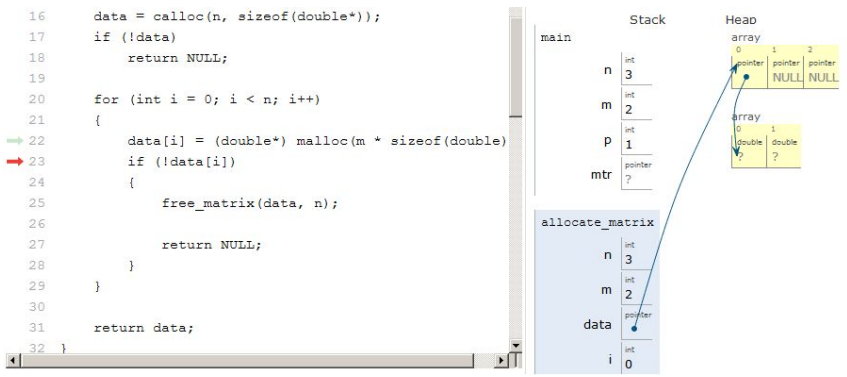
1. Перед выделением памяти



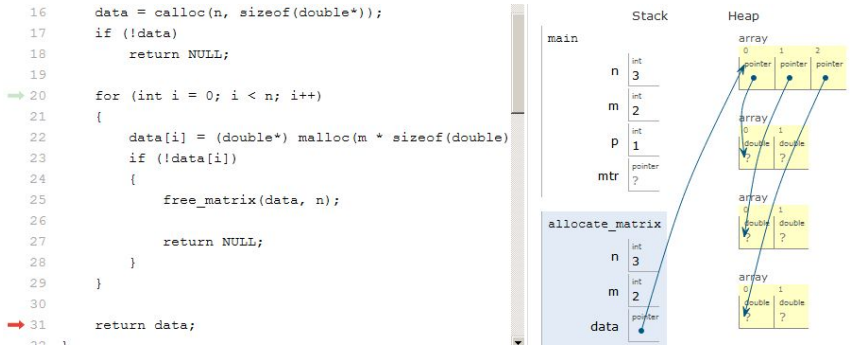
2. Выделена память под массив указателей



3. Выделена память под первую строку



4. Окончание выделения памяти



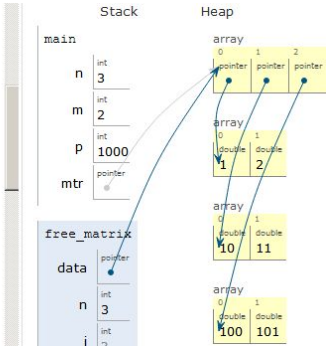
Матрица как массив указателей

```
void free_matrix(double **data, int n)
{
    for (int i = 0; i < n; i++)
       // free можно передать NULL
       free(data[i]);

    free(data);
}
```

Матрица как массив указателей

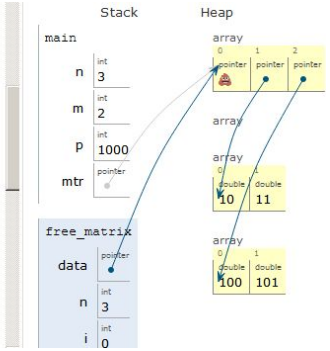
1. Перед освобождением памяти

```
    }

    return data;
}


void free_matrix(double **data, int n)
{
    for (int i = 0; i < n; i++)
        if (data[i])
            free(data[i]);

    free(data);
}
```



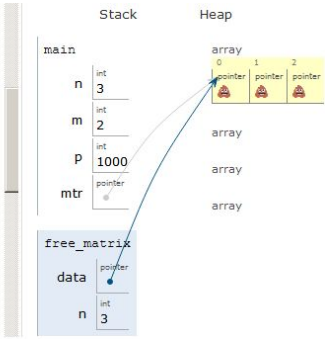
2. Освобождена память из-под первой строки

```
    }

    return data;
}


void free_matrix(double **data, int n)
{
    for (int i = 0; i < n; i++)
        if (data[i])
            free(data[i]);

    free(data);
}
```



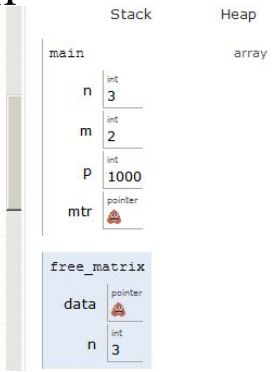
3. Освобождена память из-под строк

```
    }

    return data;
}


void free_matrix(double **data, int n)
{
    for (int i = 0; i < n; i++)
        if (data[i])
            free(data[i]);

    free(data);
}
```



4. Освобождена память из-под массива указателей

```
    }

    return data;
}


void free_matrix(double **data, int n)
{
    for (int i = 0; i < n; i++)
        if (data[i])
            free(data[i]);

    free(data);
}
```



Матрица как массив указателей

- Преимущества:
  - Возможность обмена строки через обмен указателей.
  - СПРП может отследить выход за пределы строки.

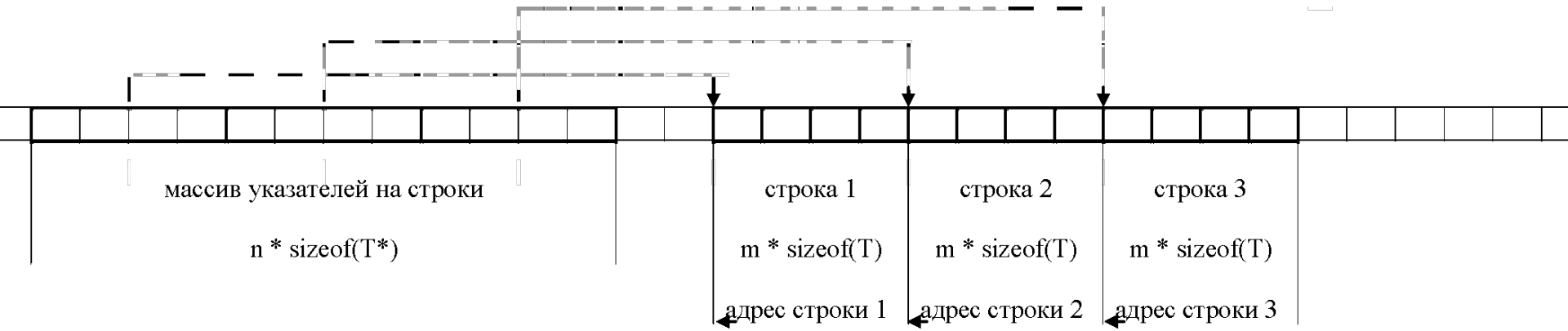
- Недостатки:
  - Сложность выделения и освобождения памяти.
  - Память под матрицу "не лежит" одним куском.

Объединение подходов (1)

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



Объединение подходов (1)

Алгоритм выделения памяти

Вход: количество строк (n) и количество столбцов (m)

Выход: указатель на массив строк матрицы (p)

- Выделить память под массив указателей на строки (p)
- Обработать ошибку выделения памяти
- Выделить память под данные (т.е. под строки, q)
- Обработать ошибку выделения памяти
- В цикле по количеству строк матрицы (0 <= i < n)
  - p[i]=адрес i-ой строки в массиве q

Объединение подходов (1)

Алгоритм освобождения памяти

Вход: указатель на массив строк матрицы (p)

- Освободить память из-под данных (адрес данных = адрес строки 0 (т.е. p[0]))
- Освободить память из-под массива указателей (p)

Объединение подходов (1)

```
double** allocate_matrix(int n, int m)
{
    double **ptrs, *data;

    ptrs = malloc(n * sizeof(double*));
    if (!ptrs)
        return NULL;

    data = malloc(n * m * sizeof(double));
    if (!data)
    {
        free(ptrs);
```

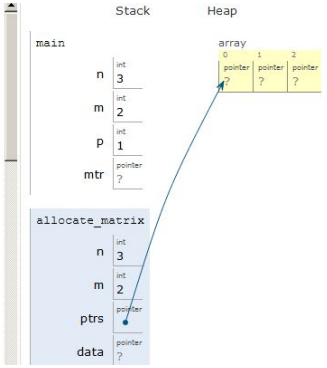
Объединение подходов (1)

```
        return NULL;
    }

    for (int i = 0; i < n; i++)
        ptrs[i] = data + i * m;

    return ptrs;
}
```

Объединение подходов (1)

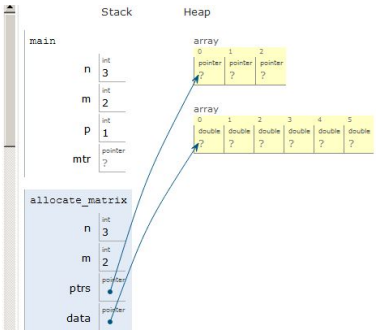
1. Выделение памяти по массив указателей

```
1  double** allocate_matrix(int n, int m)
2  {
3      double **ptrs = malloc(n * sizeof(double*));
4
5      if (ptrs)
6      {
7          double *data = malloc(n * m * sizeof(double));
8
9          for(int i = 0; i < n; i++)
10             ptrs[i] = data + i * m;
11     }
12
13     return ptrs;
14 }
15
16 void free_matrix(double **ptrs)
17 {
18     free(ptrs[0]);
```



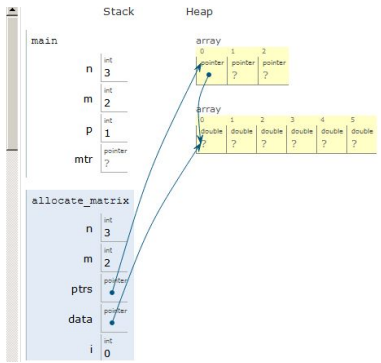
2. Выделение памяти под данные

```
1  double** allocate_matrix(int n, int m)
2  {
3      double **ptrs = malloc(n * sizeof(double*));
4
5      if (ptrs)
6      {
7          double *data = malloc(n * m * sizeof(double));
8
9          for(int i = 0; i < n; i++)
10             ptrs[i] = data + i * m;
11     }
12
13     return ptrs;
14 }
15
16 void free_matrix(double **ptrs)
17 {
18     free(ptrs[0]);
```



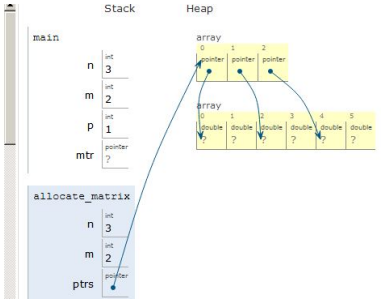
3. Вычисление адреса первой строки

```
1  double** allocate_matrix(int n, int m)
2  {
3      double **ptrs = malloc(n * sizeof(double*));
4
5      if (ptrs)
6      {
7          double *data = malloc(n * m * sizeof(double));
8
9          for(int i = 0; i < n; i++)
10             ptrs[i] = data + i * m;
11     }
12
13     return ptrs;
14 }
15
16 void free_matrix(double **ptrs)
17 {
18     free(ptrs[0]);
19
```



4. Адреса всех строк вычислены

```
1  double** allocate_matrix(int n, int m)
2  {
3      double **ptrs = malloc(n * sizeof(double*));
4
5      if (ptrs)
6      {
7          double *data = malloc(n * m * sizeof(double));
8
9          for(int i = 0; i < n; i++)
10             ptrs[i] = data + i * m;
11     }
12
13     return ptrs;
14 }
15
16 void free_matrix(double **ptrs)
```



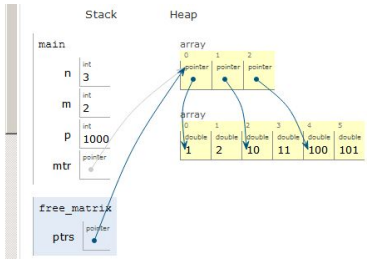
Объединение подходов (1)

```
void free_matrix(double **ptrs)
{
    free(ptrs[0]);

    free(ptrs);
}
```

Объединение подходов (1)

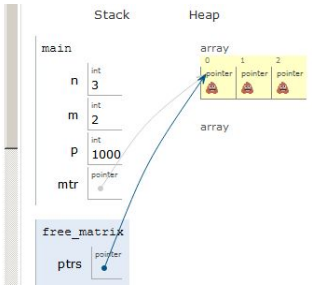
1. Перед освобождением памяти

```
10              ptrs[i] = data + i * m;
11      }
12
13      return ptrs;
14  }
15
16  void free_matrix(double **ptrs)
17  {
18      free(ptrs[0]);
19
20      free(ptrs);
21  }
22
23
```



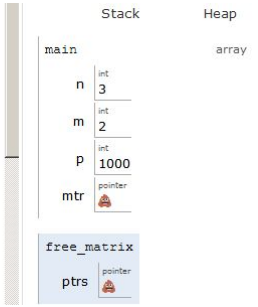
2. Память под данные освобождена

```
10              ptrs[i] = data + i * m;
11      }
12
13      return ptrs;
14  }
15
16  void free_matrix(double **ptrs)
17  {
18      free(ptrs[0]);
19
20      free(ptrs);
21  }
22
23
```



3. Память под указатели освобождена

```
10              ptrs[i] = data + i * m;
11      }
12
13      return ptrs;
14  }
15
16  void free_matrix(double **ptrs)
17  {
18      free(ptrs[0]);
19
20      free(ptrs);
21  }
22
23
```



Объединение подходов (1)

Преимущества:

- Относительная простота выделения и освобождения памяти.
- Возможность использовать как одномерный массив.
- Перестановка строк через обмен указателей.

Недостатки:

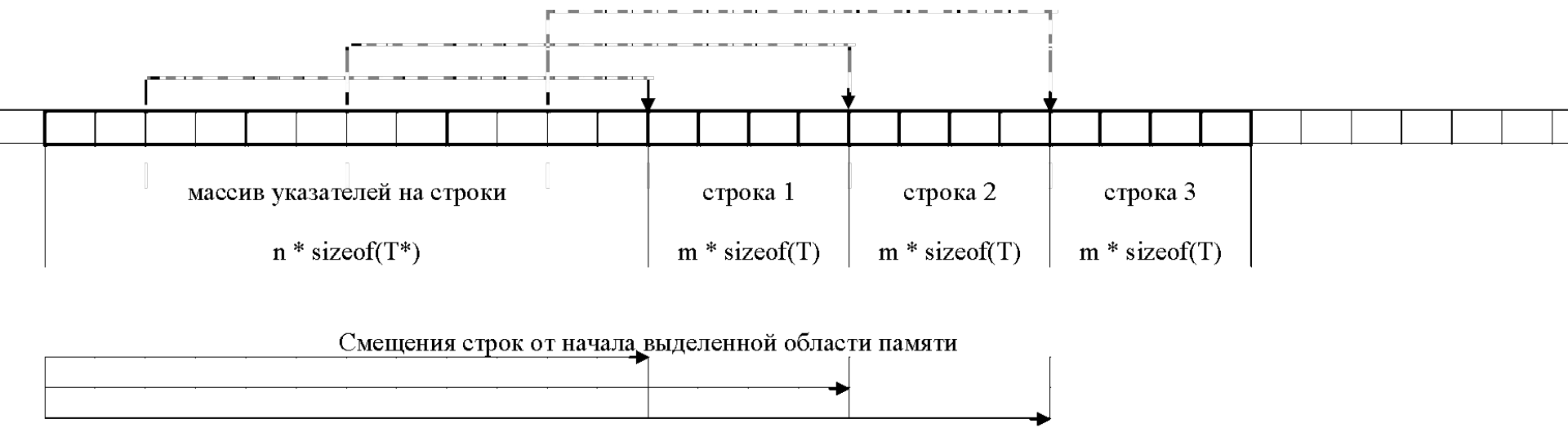
- Относительная сложность начальной инициализации.
- СПРП не может отследить выход за пределы строки.

Объединение подходов (2)

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



Объединение подходов (2)

Алгоритм выделения памяти

Вход: количество строк (n) и количество столбцов (m)

Выход: указатель на массив строк матрицы (p)

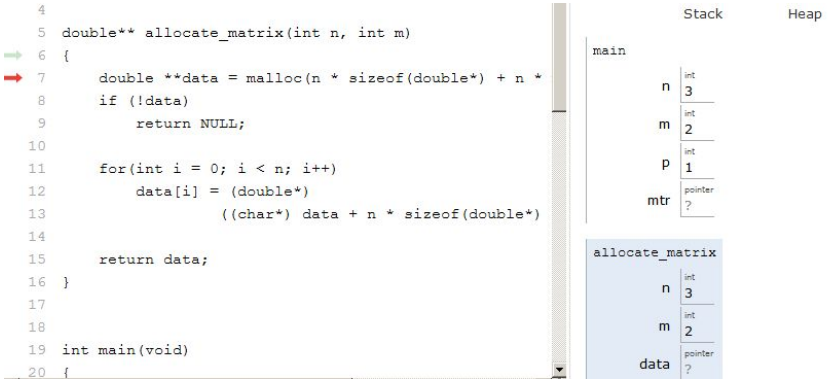
- Выделить память под массив указателей на строки и элементы матрицы (p)
- Обработать ошибку выделения памяти
- В цикле по количеству строк матрицы (0 <= i < n)
  - Вычислить адрес i-ой строки матрицы (q)
  - p[i]=q

Объединение подходов (2)

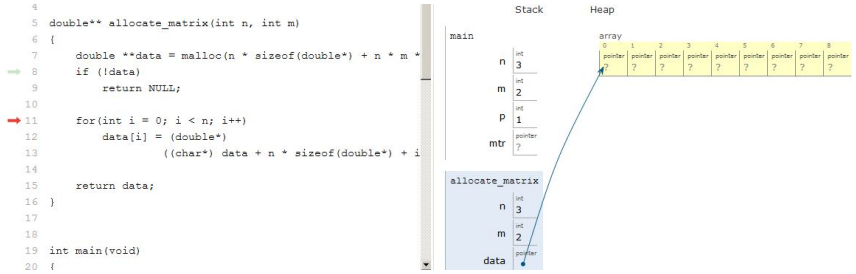
```
double** allocate_matrix_solid(int n, int m)
{
    double **data = malloc(n * sizeof(double*) +
                                n * m * sizeof(double));
    if (!data)
        return NULL;

    for (int i = 0; i < n; i++)
        data[i] = (double*)((char*) data +
                        n * sizeof(double*) +
                                    i * m *
  sizeof(double));
    return data;
}
```

Объединение подходов (2)

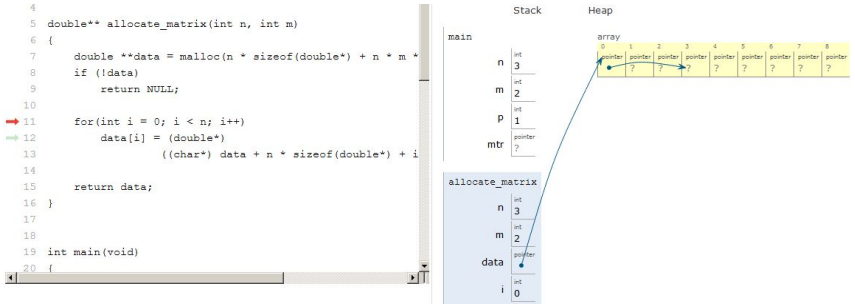
1. Перед выделением памяти



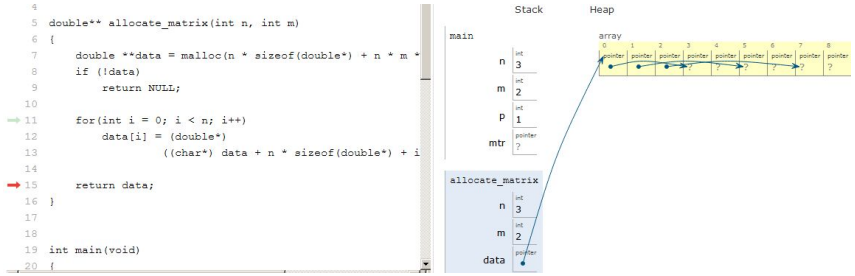
2. Выделение памяти



3. Вычисление адреса первой строки



4. Адреса всех строк вычислены



Объединение подходов (2)

Преимущества:

- Простота выделения и освобождения памяти.
- Возможность использовать как одномерный массив.
- Перестановка строк через обмен указателей.

Недостатки:

- Сложность начальной инициализации.
- СПРП не может отследить выход за пределы строки.

Литература

1. http://c-faq.com/aryptr/dynmuldimary.html

29