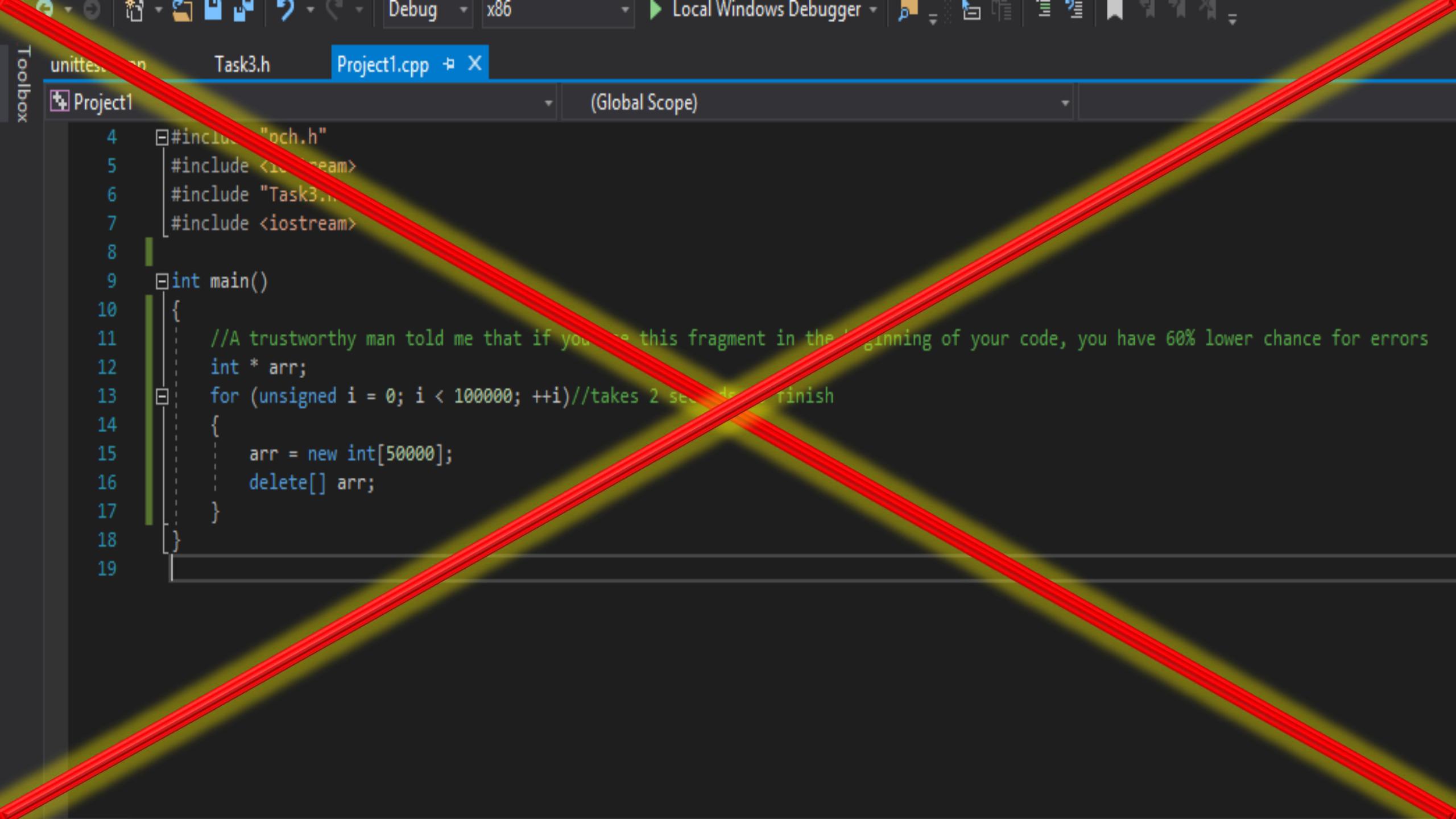


Работа с динамична памет

- Работата с динамична памет включва:
 1. Заделяне на такава памет
 2. Обработка на данни
 3. Освобождаване на заделената памет
- Като цяло 2. е optional, но реално ние използваме динамична памет точно заради 2.



Заделяне на динамична памет

- За заделяне на динамична памет се използват операторите:
 - `new <тип> [<стойност>]` – заделя памет за точно един нов обект, инициализира го и връща поинтър към него
 - `new <тип>[<число n>]` – заделя памет за n-мерна редица, инициализира всички обекти в нея и връща поинтър към първия
- Примери:
 - `char * dChar = new char;`
 - `char * dCharA = new char('A');`
 - `char * dCharArr = new char[6];`

Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

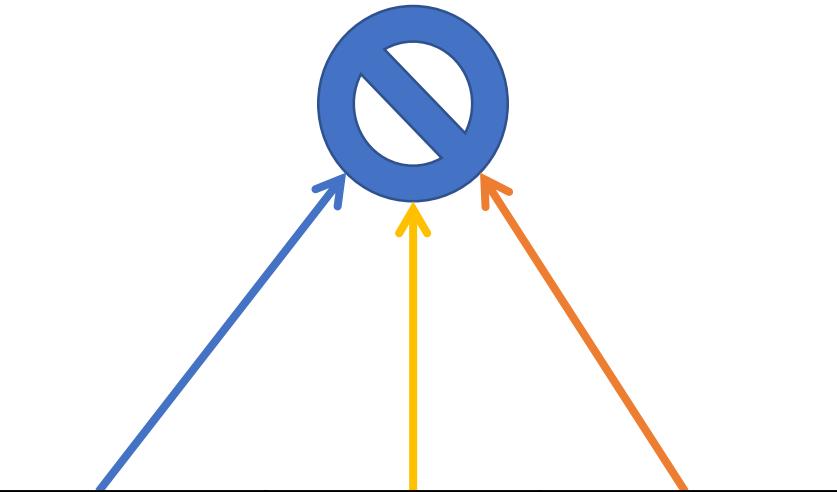
Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;
```

```
dChar = new char;
```

```
dCharA = new char('A');
```

```
dCharArr = new char[6];
```

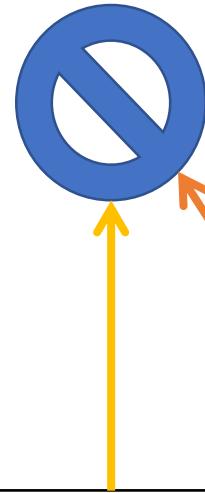


0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			undefined	undefined	undefined
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

Заделяне на динамична памет - визуализация

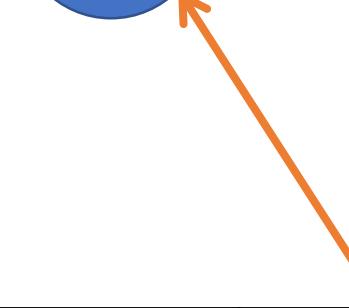
```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	undefined	undefined
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17



Заделяне на динамична памет - визуализация

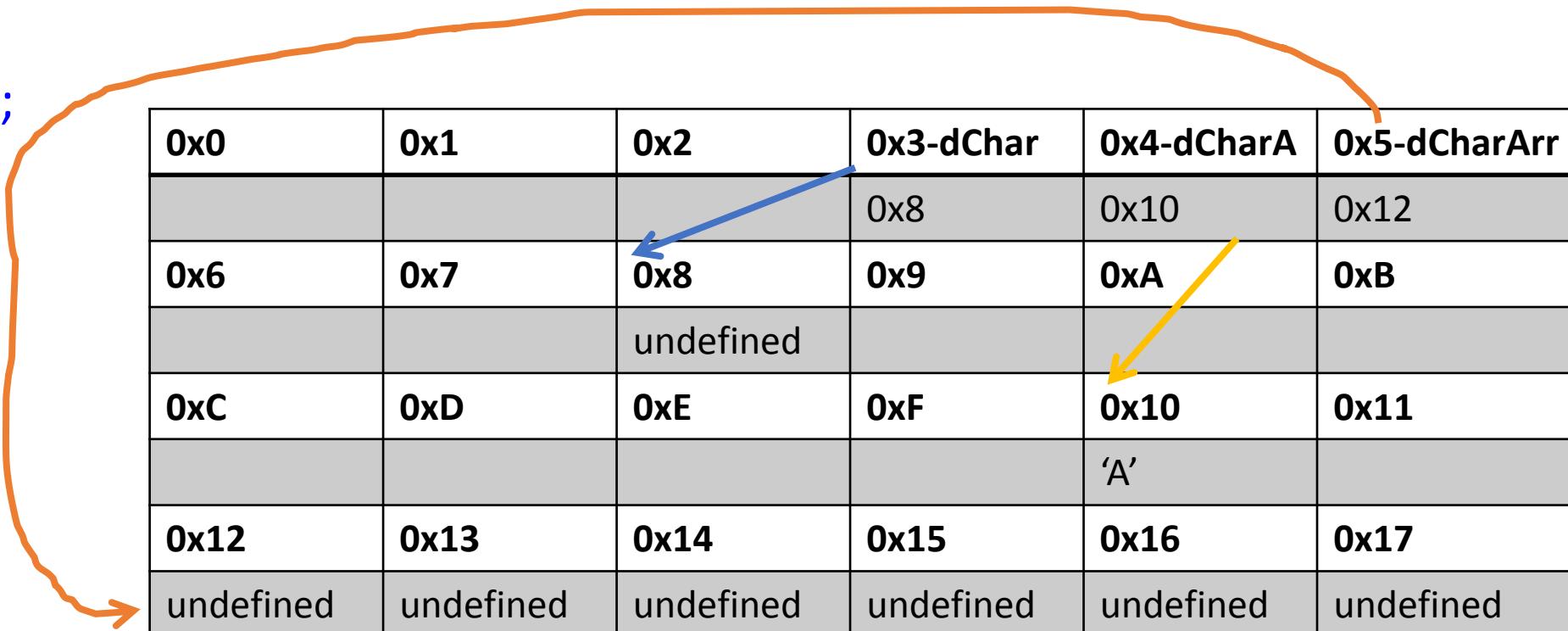
```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```



0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	undefined
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17

Заделяне на динамична памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];
```



Освобождаване на заделена памет

- За освобождаване на динамична памет се използват операторите:
 - `delete <адрес>`
 - `delete[] <адрес>`
- Примери(спрямо предишните примери):
 - `delete dChar;`
 - `delete dCharA;`
 - `delete[] dCharArr;`

Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
		undefined			
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

The diagram illustrates the state of memory after the deallocation of pointers. A red oval encloses the first four columns of the table, representing the heap. The fifth column shows the current state of each pointer:

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
				'A'	
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

A blue arrow points from the value 0x8 in the fourth column to the cell at address 0x2. A yellow arrow points from the value 0x10 in the fifth column to the cell at address 0xF. The addresses 0x2, 0x8, 0x9, 0xA, 0x10, and 0xF are highlighted in blue.

Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17
undefined	undefined	undefined	undefined	undefined	undefined

Освобождаване на заделена памет - визуализация

```
char * dChar, * dCharA, * dCharArr;  
dChar = new char;  
dCharA = new char('A');  
dCharArr = new char[6];  
delete dChar;  
delete dCharA;  
delete[] dCharArr;
```

The diagram illustrates the state of memory after the deallocation of pointers. A red oval encloses the first four rows of the memory grid, representing the heap. The last two rows represent the stack.

The memory grid shows the following values:

0x0	0x1	0x2	0x3-dChar	0x4-dCharA	0x5-dCharArr
			0x8	0x10	0x12
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

Annotations indicate the state of pointers:

- A blue arrow points from the label **0x3-dChar** to the cell containing **0x8**, which is highlighted in blue.
- A yellow arrow points from the label **0x4-dCharA** to the cell containing **0xA**, which is highlighted in yellow.
- A yellow arrow points from the label **0x5-dCharArr** to the cell containing **0x10**, which is highlighted in yellow.

Освобождаване на заделена памет

- `delete` операторите могат да освобождават само динамично заделена памет
- Не е позволено освобождаването на стекова памет:

```
int a;  
int* b = &a;  
delete b;
```

- Не е позволено частично освобождаване на памет:

```
int* a = new int[10];  
int * b = a+1;  
delete[] b;
```

Освобождаване на заделена памет

- За ваше улеснение, може да използвате правилото:
 - new => delete
 - new[] => delete[] //delete[] си знае колко памет трябва да освободи
- След освобождаването на дадена памет, тя става недостъпна и обръщането към нея обикновено води до фойерверки
- delete и delete[] върху nullptr са безобидни и не правят нищо
- Винаги освобождавайте паметта, която сте заделили!!!

Освобождаване на заделена памет

- standard (5.3.5/2) :
 - In the first alternative (delete object), the value of the operand of delete shall be a pointer to a non-array object or a pointer to a sub-object (1.8) representing a base class of such an object (clause 10). **If not, the behavior is undefined.**
 - In the second alternative (delete array), the value of the operand of delete shall be the pointer value which resulted from a previous array new-expression. **If not, the behavior is undefined.**

Обработка на динамични данни

- Особеностите на динамичните данни са:
 - заделяне на памет и освобождаването ѝ се извършва по време на изпълнение
 - липса на име
- Всичко останало си е както и преди, като трябва:
 - да се съобрази, че се обръщаме към пойнтър от дадения тип, а не просто към обект
 - от сходствата между пойнтъри и масиви следва, че можем спокойно да използваме оператор [] //припомните си какво разгледахме там

Обработка на динамични данни

- Възможността да контролираме кога дадена памет да бъде освободена ни дава изненадващо много нови възможности
- Вече функция, връщаща пойнтър, може да има много повече приложения от преди
- With Great Power Comes Great Responsibility!

Обработка на динамични данни

- Как се създава матрица NxM ?

```
int ** Matrix (const unsigned n, const unsigned m)
{
    int ** tmp = new int * [n];
    for(unsigned i = 0; i<n; ++i)
        tmp[i] = new int [m];
    return tmp;
}
```

Обработка на динамични данни

- Как се изтрива матрица NxM ?

```
int ** A = Matrix(3,2);
for(unsigned i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3;++)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9	0xA	0xB
	10				
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
		10	0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		Undefined	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
	10		0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	0	Undefined	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{  
    bool ** tmp = new bool * [n];  
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];  
    return tmp;  
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
				11	
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	0	0x12	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined				

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
		10	0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	1	0x12	Undefined	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined				

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
				11	
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	1	0x12	0x14	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined		

The diagram illustrates the memory layout for the `Matrix` function. It shows a grid of memory addresses and their corresponding values. The columns are labeled with addresses: 0x0 - n, 0x1, 0x2, 0x3, 0x4 - m, and 0x5. The rows are labeled with addresses: 0x6, 0x7, 0x8, 0x9 - tmp, 0xA, and 0xB. The value 11 is at 0x0. The value 10 is at 0x6. The value 0xE is at 0x11. The value 0xF is at 0x10. The value 0x12 is at 0xC. The value 0x13 is at 0x12. The value 0x14 is at 0x14. The value 0x15 is at 0x15. The value 0x16 is at 0x16. The value 0x17 is at 0x17. There are three colored arrows: a blue arrow pointing from 0xE at 0x11 to 0xF at 0x10; an orange arrow pointing from 0x12 at 0xC to 0x13 at 0x12; and a green arrow pointing from 0x14 at 0x14 to 0x15 at 0x15.

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
	10		0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	10	0x12	0x14	Undefined	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined		

```
Diagram illustrating the memory layout and pointer assignments for the Matrix function. The memory is organized into a grid of cells. Row 0: 0x0 - n, 0x1, 0x2, 0x3 (containing 11), 0x4 - m, 0x5. Row 1: 0x6, 0x7, 0x8, 0x9 - tmp (containing 0xE), 0xA, 0xB. Row 2: 10, 0xE, 0xF, 0x10, 0x11. Row 3: 10, 0x12, 0x14, Undefined, 0x16, 0x17. Row 4: 0x12, 0x13, 0x14, 0x15, 0x16, 0x17. Row 5: Undefined, Undefined, Undefined, Undefined, ., . Cells 0x9 - tmp, 0x12, and 0x14 are highlighted with colors (blue, orange, green) and connected by arrows to their respective source values in the code.
```

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{  
    bool ** tmp = new bool * [n];  
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];  
    return tmp;  
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
				11	
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
			0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	10	0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

```
graph LR; 11[11] --> 0x9["0x9 - tmp"]; 10[10] --> 0x12["0x12"]; 0xE[0xE] --> 0x14["0x14"]; 0x16[0x16] --> 0x16["0x16"];
```

Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{
```

```
    bool ** tmp = new bool * [n];
```

```
    for(char i = 0; i<n; ++i)
```

```
        tmp[i] = new bool [m];
```

```
    return tmp;
```

```
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
			11		
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
	10		0xE		
0xC	0xD - i	0xE	0xF	0x10	0x11
	11	0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

```
graph LR; E1[0xE] -- Blue --> E2[0xE]; E3[0x12] -- Orange --> E4[0x13]; E5[0x14] -- Green --> E6[0x15]; E7[0x16] -- Yellow --> E8[0x16]
```

Обработка на динамични данни - визуализация

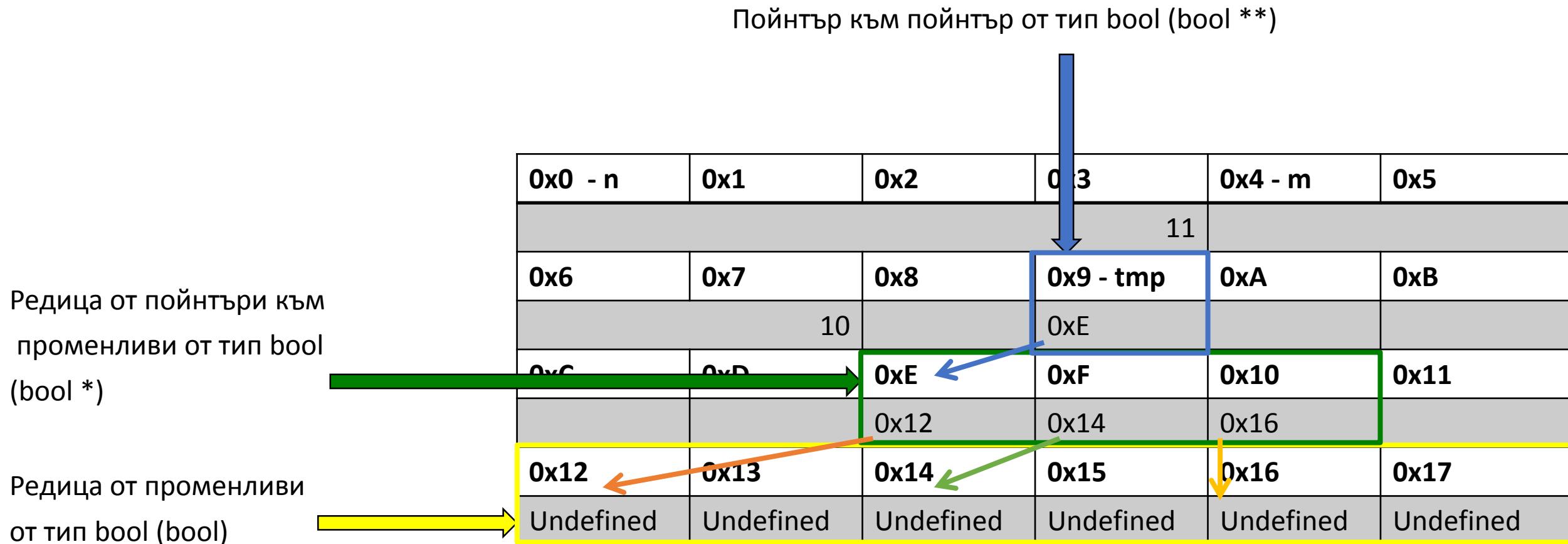
```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{  
    bool ** tmp = new bool * [n];  
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];  
    return tmp;  
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
				11	
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
			0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

```
graph TD; 0x9 --> 0xE; 0xC --> 0x13; 0xD --> 0x14; 0xE --> 0x16;
```

Обработка на динамични данни - визуализация



Обработка на динамични данни - визуализация

```
bool ** Matrix (const unsigned n, const unsigned m)
```

```
{  
    bool ** tmp = new bool * [n];  
    for(char i = 0; i<n; ++i)  
        tmp[i] = new bool [m];  
    return tmp;  
}
```

0x0 - n	0x1	0x2	0x3	0x4 - m	0x5
				11	
0x6	0x7	0x8	0x9 - tmp	0xA	0xB
			0xE		
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

```
graph TD; 0x9 --> 0xE; 0xC --> 0x13; 0xD --> 0x14; 0xE --> 0x16;
```

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
```

```
for(char i =0; i<3; ++i)
```

```
{
```

```
    delete[] A[i];
```

```
}
```

```
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates the memory layout and deallocation process for a 3x2 matrix. The matrix is initially allocated at address 0xB - A. It consists of six cells: 0x0, 0x1, 0x2, 0x3, 0x4, and 0x5. The first row contains 0x6, 0x7, 0x8, 0x9, 0xA, and 0xB - A. The second row contains 0xC, 0xD, 0xE, 0xF, 0x10, and 0x11. The third row contains 0x12, 0x13, 0x14, 0x15, 0x16, and 0x17. The code then iterates through the matrix and calls delete[] A[i] for each row i. This results in the deallocation of the first two rows. The remaining two rows (0xC-E and 0x12-15) are shown as undefined. Arrows indicate the flow of deallocation: a blue arrow from 0xE to 0xF, an orange arrow from 0x12 to 0x13, a green arrow from 0x14 to 0x15, and a yellow arrow from 0x16 to 0x17.

Обработка на динамични данни - визуализация

Матрица 3X2

0x12	0x13
0x14	0x15
0x16	0x17

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	0
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a 6x6 grid of memory cells, each containing a hexadecimal value. The first five rows represent the stack, and the bottom row represents the heap.

- Row 1:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5
- Row 2:** 0x6, 0x7, 0x8, 0x9, 0xA, 0xB - A
- Row 3:** 0xC, 0xD, 0xE, 0xF, 0x10, 0x11 - i
- Row 4:** (empty), (empty), 0x12, 0x14, 0x16, 0
- Row 5:** 0x12, 0x13, 0x14, 0x15, 0x16, 0x17
- Row 6:** Undefined, Undefined, Undefined, Undefined, Undefined, Undefined

Annotations with arrows indicate specific memory locations:

- A blue arrow points from the cell containing **0xB - A** to the cell containing **0x10**.
- An orange arrow points from the cell containing **0x12** to the cell containing **0x13**.
- A green arrow points from the cell containing **0x14** to the cell containing **0x15**.
- A yellow arrow points from the cell containing **0x16** to the cell containing **0x16**.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	0
0x12	0x13	0x14	0x15	0x16	0x17
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a 6x6 grid of memory cells, each containing a hexadecimal value. The first five rows represent the stack, and the bottom row represents the heap.

- Row 1:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5
- Row 2:** 0x6, 0x7, 0x8, 0x9, 0xA, 0xB - A
- Row 3:** 0xC, 0xD, 0xE, 0xF, 0x10, 0x11 - i
- Row 4:** (empty), (empty), 0x12, 0x14, 0x16, 0
- Row 5:** 0x12, 0x13, 0x14, 0x15, 0x16, 0x17
- Row 6:** Undefined, Undefined, Undefined, Undefined, Undefined, Undefined

Annotations with arrows indicate specific memory locations:

- A blue arrow points from the cell containing **0xB - A** to the cell containing **0x10**.
- An orange arrow points from the cell containing **0x12** to the cell containing **0x13**.
- A green arrow points from the cell containing **0x14** to the cell containing **0x15**.
- A yellow arrow points from the cell containing **0x16** to the cell containing **0x16**.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
					0
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a 6x6 grid of memory cells, each containing a hexadecimal value. The columns are indexed from 0x0 to 0x5, and the rows are indexed from 0x0 to 0x17. A blue arrow points from cell 0xA to cell 0xE, indicating the deallocation of row 0x11. An orange arrow points from cell 0x12 to cell 0x13, indicating the deallocation of row 0x12. A green arrow points from cell 0x14 to cell 0x15, indicating the deallocation of row 0x14. A yellow arrow points from cell 0x16 to cell 0x16, indicating the deallocation of row 0x16. Cells containing 'Undefined' represent free memory.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a 6x6 grid of memory cells, each containing a hexadecimal value. The columns are indexed from 0x0 to 0x5, and the rows are indexed from 0x0 to 0x5. The values in the grid are as follows:

- Row 0: 0x0, 0x1, 0x2, 0x3, 0x4, 0x5
- Row 1: 0x6, 0x7, 0x8, 0x9, 0xA, 0xB - A
- Row 2: 0xC, 0xD, 0xE, 0xF, 0x10, 0x11 - i
- Row 3: (empty)
- Row 4: 0x12, 0x13, 0x12, 0x14, 0x16, 1
- Row 5: 0x12, 0x13, 0x14, 0x15, 0x16, 0x17
- Row 6: (empty)

Annotations with arrows indicate specific memory locations:

- A blue arrow points from the cell at index 0xB (containing 0xB) to the cell at index 0x10 (containing 0x10).
- An orange arrow points from the cell at index 0x12 (containing 0x12) to the cell at index 0x13 (containing 0x13).
- A green arrow points from the cell at index 0x14 (containing 0x14) to the cell at index 0x15 (containing 0x15).
- A yellow arrow points from the cell at index 0x16 (containing 0x16) to the cell at index 0x16 (containing 0x16).

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
		Undefined	Undefined	Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a grid of memory cells with addresses from 0x0 to 0x17. The first two rows (0x0-0x5) are entirely undefined. Row 3 contains values 0x6 through 0xB. Row 4 contains 0xE. Row 5 contains 0xC through 0x11. Row 6 contains 0x12 through 0x16. Row 7 contains 0x17. Arrows indicate the movement of pointers: a blue arrow points from 0x11 to 0x16, an orange arrow points from 0x12 to 0x13, a green arrow points from 0x14 to 0x15, and a yellow arrow points from 0x16 to 0x16.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	1
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates the state of memory after the deallocation of matrix A. It shows a 6x6 grid of memory cells, each containing a hexadecimal value. The columns are indexed from 0x0 to 0x5, and the rows are indexed from 0x0 to 0x17. A blue arrow points from cell 0x11 (containing 0xA) to cell 0xB (containing '0xB - A'). An orange arrow points from cell 0x12 (containing 0xC) to cell 0x13 (containing 0xD). A green arrow points from cell 0x14 (containing 0xE) to cell 0x15 (containing 0xF). A yellow arrow points from cell 0x16 (containing 0x11) to cell 0x16 (containing '1'). Cells 0x12, 0x13, 0x14, 0x15, and 0x16 are highlighted with orange backgrounds, indicating they are part of the deallocated matrix A.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
					10
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates the state of memory after the deallocation of matrix A. It shows a 6x6 grid of memory cells. The first five columns represent pointers to dynamically allocated arrays, while the last column contains their respective values.

- Row 1:** Contains pointers to the first row of each array: 0x0, 0x1, 0x2, 0x3, 0x4, and 0x5.
- Row 2:** All cells are grayed out, indicating they are no longer valid.
- Row 3:** Contains pointers to the second row: 0x6, 0x7, 0x8, 0x9, 0xA, and 0xB - A.
- Row 4:** All cells are grayed out.
- Row 5:** Contains pointers to the third row: 0xC, 0xD, 0xE, 0xF, 0x10, and 0x11 - i.
- Row 6:** All cells are grayed out.
- Row 7:** Contains the actual values stored at those addresses: 0x12, 0x13, 0x14, 0x15, 0x16, and 0x17.
- Row 8:** All cells are grayed out.

Annotations with arrows highlight specific elements:

- An orange arrow points from the value 0x12 in Row 7 to the cell 0x13 in Row 7.
- A green arrow points from the value 0x14 in Row 7 to the cell 0x15 in Row 7.
- A blue arrow points from the value 0xE in Row 5 to the cell 0xF in Row 5.
- A yellow arrow points from the value 0x16 in Row 7 to the cell 0x16 in Row 8.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
0xC	0xD	0xE	0xF	0x10	0x11 - i
0x12	0x13	0x14	0x15	0x16	10
0x12	0x13	0x14	0x15	0x16	0x17
				Undefined	Undefined

The diagram illustrates the state of memory after the execution of the provided C++ code. It shows a grid of memory addresses (0x0 to 0x17) and their corresponding values.

- Cells 0x0 through 0x4 are empty (gray).
- Cell 0x5 contains the value 0xB - A.
- Cells 0x6 through 0xA contain empty memory blocks (gray).
- Cell 0xB contains the value 0x11 - i.
- Cells 0xC through 0xF contain empty memory blocks (gray).
- Cell 0x10 contains the value 10.
- Cells 0x11 through 0x15 contain empty memory blocks (gray).
- Cells 0x16 and 0x17 are undefined (gray).

Annotations with arrows indicate specific memory locations:

- A blue arrow points from the value 0xB - A at address 0xA to the cell at 0x11-i.
- An orange arrow points from the value 0x12 at address 0xC to the cell at 0x13.
- A green arrow points from the value 0x14 at address 0xD to the cell at 0x15.
- A yellow arrow points from the value 0x16 at address 0xE to the cell at 0x16.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
		0x12	0x14	0x16	10
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates the memory layout and the effect of deallocation on pointers. It shows a grid of memory addresses (0x0 to 0x17) and their corresponding values.

- Initial State:** Addresses 0x0 through 0x4 are empty. Addresses 0x6 through 0xA contain values 0x6, 0x7, 0x8, 0x9, and 0xA respectively. Address 0xB contains the value "A". Address 0xC contains the value "0xC". Address 0xD contains the value "0xD". Address 0xE contains the value "0xE". Address 0xF contains the value "0xF". Address 0x10 contains the value "0x10". Address 0x11 contains the value "0x11 - i". Address 0x12 contains the value "0x12". Address 0x13 contains the value "0x13". Address 0x14 contains the value "0x14". Address 0x15 contains the value "0x15". Address 0x16 contains the value "0x16". Address 0x17 contains the value "0x17".
- After Deallocation:** The code `delete[] A[i];` dealslocate the row `A[i]` at address 0xB. This results in:
 - Address 0xB now contains a null pointer (empty).
 - Address 0xC now contains the value "0x12" (the previous value of 0xE).
 - Address 0xD now contains the value "0x13" (the previous value of 0xE).
 - Address 0xE now contains the value "0x14" (the previous value of 0xE).
 - Address 0xF now contains the value "0x15" (the previous value of 0xE).
 - Address 0x10 now contains the value "0x16" (the previous value of 0xE).
 - Address 0x11 now contains the value "10" (the previous value of 0xE).
 - Address 0x12 now contains the value "0x12" (the previous value of 0xE).
 - Address 0x13 now contains the value "0x13" (the previous value of 0xE).
 - Address 0x14 now contains the value "0x14" (the previous value of 0xE).
 - Address 0x15 now contains the value "0x15" (the previous value of 0xE).
 - Address 0x16 now contains the value "0x16" (the previous value of 0xE).
 - Address 0x17 now contains the value "0x17" (the previous value of 0xE).

Arrows indicate the movement of pointers from the deallocated row (0xB) to the next row (0xC), and from the deallocated row (0xB) to the next row (0xC). The colors of the arrows correspond to the colors of the arrows in the original image.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11 - i
					11
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates the memory layout and deallocation process. It shows a 6x6 grid of memory cells. The first five columns are labeled with addresses 0x0 through 0x4, and the last column is labeled with values 0xB - A, 0xE, 0x11 - i, 11, and 0x17. The first five rows are labeled with addresses 0x0 through 0x4, and the last row is grayed out. A blue arrow points from cell 0xB to the row containing 0x11 - i. An orange arrow points from cell 0x12 to cell 0x13. A green arrow points from cell 0x14 to cell 0x15. A yellow arrow points from cell 0x16 to the bottom row.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
0xC	0xD	0xE	0xF	0x10	0x11
		0x12	0x14	0x16	0x17
0x12	0x13	0x14	0x15	0x16	0x17

The diagram illustrates the memory layout and deallocation process. It shows a grid of memory cells with addresses in hex. The first row contains 0x0, 0x1, 0x2, 0x3, 0x4, and 0x5. The second row contains 0x6, 0x7, 0x8, 0x9, 0xA, and 0xB - A. The third row contains 0xC, 0xD, 0xE, 0xF, 0x10, and 0x11. The fourth row contains empty cells. The fifth row contains 0x12, 0x13, 0x14, 0x15, 0x16, and 0x17. The sixth row contains 0x12, 0x13, 0x14, 0x15, 0x16, and 0x17. Arrows indicate the deallocation of elements: a blue arrow from 0xB to 0xE, an orange arrow from 0x12 to 0x13, a green arrow from 0x14 to 0x15, and a yellow arrow from 0x16 to the 0x16 in the fifth row.

Обработка на динамични данни - визуализация

```
bool ** A = Matrix(3,2);
for(char i =0; i<3; ++i)
{
    delete[] A[i];
}
delete[] A;
```

0x0	0x1	0x2	0x3	0x4	0x5
0x6	0x7	0x8	0x9	0xA	0xB - A
					0xE
0xC	0xD	0xE	0xF	0x10	0x11
0x12	0x13	0x14	0x15	0x16	0x17

A diagram illustrating the memory layout and deallocation of a 3x2 matrix. The matrix is represented as a grid of 6 cells. The first two rows are entirely grayed out, representing freed memory. The third row contains values: 0x6, 0x7, 0x8, 0x9, 0xA, and 0xB - A. A blue arrow points from the value 0xE in the third row to the cell containing 0xF in the fourth row, indicating the deallocation of the third matrix row.