

Указатели. Динамическое распределение памяти

Старший преподаватель
Шпаков Сергей Андреевич

Кафедра информатики
и информационных технологий

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

- ❖ **Бит** — это наименьший элемент компьютерной памяти, способная хранить либо **0**, либо **1**. На физическом уровне это соответствует электрическому напряжению, которое либо есть в цепи, либо нет.
- ❖ Код программы и данные, которыми программа манипулирует, записываются в память компьютера в виде последовательности битов. Посмотрев на содержимое памяти компьютера, можно увидеть что-то вроде:

00011011011100010110010000111011...

- ❖ Бит — очень маленькая единица. В современных компьютерных системах минимальный адресуемый блок информации — **байт**.

1 байт = 8 бит

1 Кбайт	1024 байт
1 Мбайт	1024 Кбайт
1 Гбайт	1024 Мбайт
1 Тбайт	1024 Гбайт

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

- ❖ **Машинное слово** (группа байт) определяет следующие характеристики аппаратной платформы:
 - *разрядность данных*, обрабатываемых *процессором*;
 - разрядность адресуемых данных (разрядность *шины данных*);
 - максимальное значение *беззнакового целого типа*, напрямую поддерживаемого процессором: если результат арифметической операции превосходит это значение, то происходит *переполнение*;
 - максимальный объём *оперативной памяти*, напрямую адресуемой процессором.
- ❖ Слово является **машинно-зависимым** (разная длина в различных операционных системах и средах):
 - 16-битные системы и среды (1 слово = 2 байт = 16 бит);
 - 32-битные системы и среды (1 слово = 4 байт = 32 бит);
 - 64-битные системы и среды (1 слово = 8 байт = 64 бит).

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

- ❖ **Модель данных** — соотношения размерностей типов, принятых в рамках среды разработки.
- ❖ Для одной операционной системы могут существовать несколько средств разработки, придерживающихся разных моделей данных.
- ❖ Обычно преобладает только одна модель, наиболее соответствующая аппаратной и программной среде.

Тип данных / Название модели	short	int	long	ptr	long long	Примеры использования
ILP32	16	32	32	32	. . .	IBM 370; VAX Unix; many workstations
ILP32LL (or ILP32LL64)	16	32	32	32	64	Microsoft Win32; Amdahl; Convex; 1990 Unix systems; Like IP16L32; for same reason; multiple instructions for long long
LLP64 (or IL32LLP64 or P64)	16	32	32	64	64	Microsoft Win64 (x64 / IA64)
LP64 (or I32LP64)	16	32	64	64	64	Most Unix systems (Linux, Solaris, DEC OSF/1 Alpha, SGI Irix, HP UX 11)
ILP64	16	64	64	64	64	HAL; logical analog of ILP32

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

❖ **ILP32LL** — модель данных в языках C/C++ в 32-битных системах.

Тип данных / Название модели	short	int	long	ptr	long long	Примеры использования
ILP32	16	32	32	32	. . .	IBM 370; VAX Unix; many workstations
ILP32LL (or ILP32LL64)	16	32	32	32	64	Microsoft Win32; Amdahl; Convex; 1990 Unix systems; Like IP16L32; for same reason; multiple instructions for long long
LLP64 (or IL32LLP64 or P64)	16	32	32	64	64	Microsoft Win64 (x64 / IA64)
LP64 (or I32LP64)	16	32	64	64	64	Most Unix systems (Linux, Solaris, DEC OSF/1 Alpha, SGI Irix, HP UX 11)
ILP64	16	64	64	64	64	HAL; logical analog of ILP32

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

❖ **LLP64** — модель данных в языках C/C++ в 64-битных системах.

Тип данных / Название модели	short	int	long	ptr	long long	Примеры использования
ILP32	16	32	32	32	. . .	IBM 370; VAX Unix; many workstations
ILP32LL (or ILP32LL64)	16	32	32	32	64	Microsoft Win32; Amdahl; Convex; 1990 Unix systems; Like IP16L32; for same reason; multiple instructions for long long
LLP64 (or IL32LLP64 or P64)	16	32	32	64	64	Microsoft Win64 (x64 / IA64)
LP64 (or I32LP64)	16	32	64	64	64	Most Unix systems (Linux, Solaris, DEC OSF/1 Alpha, SGI Irix, HP UX 11)
ILP64	16	64	64	64	64	HAL; logical analog of ILP32

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

- ❖ Оператор **sizeof()**, получив в качестве аргумента какой-либо объект, возвращает целое число, показывающее количество байт, занимаемых этим объектом.

```
1  #include <iostream>
2  | #include <windows.h>
3
4  using namespace std;
5
6  int main() {
7      SetConsoleCP(1251);
8      SetConsoleOutputCP(1251);
9      char caption[] = "Data Type's Sizes";
10     SetConsoleTitle(caption);
11
12     int m = 0; // 4байта
13     cout << "int m = 0;\t\t"
14          << "sizeof(m) = " << sizeof(m) << " byte(s)." << endl << endl;
15
16     double k = 1.275e-5; // 8 байт
17     cout << "double k = 1.275e-5;\t"
18          << "sizeof(k) = " << sizeof(k) << " byte(s)." << endl << endl;
19
```

8. Указатели. Динамическое распределение памяти

8.1. Модель данных

```
20 char *pchar = nullptr; // 4 байта
21 cout << "char *pchar = nullptr;\t"
22     << "sizeof(pchar) = " << sizeof(pchar) << " byte(s)." << endl;
23 cout << "\t\t\t" // 1 байт
24     << "sizeof(*pchar) = " << sizeof(*pchar) << " byte(s)." << endl << endl;
25
26 short arr[6][5]; // 60 байт (6*5*2 байт)
27 cout << "short arr[6][5];\t" // 2 байта занимает один элемент в массиве
28     << "sizeof(arr[0][0]) = " << sizeof(arr[0][0]) << " byte(s)." << endl;
29 cout << "\t\t\t"
30     << "sizeof(arr) = " << sizeof(arr) << " byte(s)." << endl;
31 cout << "\t\t\t" // 30 элементов в массиве
32     << "sizeof(arr)/sizeof(arr[0][0]) = "
33     << sizeof(arr)/sizeof(arr[0][0]) << " element(s)." << endl;
34
35 cout << "\t\t\t" // 6 по 5*2 байт = 10 байт занимает каждая строка массива
36     << "sizeof(arr[0]) = " << sizeof(arr[0]) << " byte(s)." << endl;
37 cout << "\t\t\t" // 6 строк в массиве
38     << "sizeof(arr)/sizeof(arr[0]) = "
39     << sizeof(arr)/sizeof(arr[0]) << " line(s)." << endl;
40 cout << "\t\t\t" // 5 элементов в каждой строке
41     << "sizeof(arr[0])/sizeof(arr[0][0]) = "
42     << sizeof(arr[0])/sizeof(arr[0][0]) << " elements(s)." << endl;
43
44 return 0;
45 }
```


8. Указатели. Динамическое распределение памяти

8.1. Модель данных

```
Data Type's Sizes
int m = 0;          sizeof(m) = 4 byte(s).

double k = 1.275e-5;  sizeof(k) = 8 byte(s).

char *pchar = nullptr;  sizeof(pchar) = 4 byte(s).
                        sizeof(*pchar) = 1 byte(s).

short arr[6][5];    sizeof(arr[0][0]) = 2 byte(s).
                    sizeof(arr) = 60 byte(s).
                    sizeof(arr)/sizeof(arr[0][0]) = 30 element(s).
                    sizeof(arr[0]) = 10 byte(s).
                    sizeof(arr)/sizeof(arr[0]) = 6 line(s).
                    sizeof(arr[0])/sizeof(arr[0][0]) = 5 elements(s).

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

8. Указатели. Динамическое распределение памяти

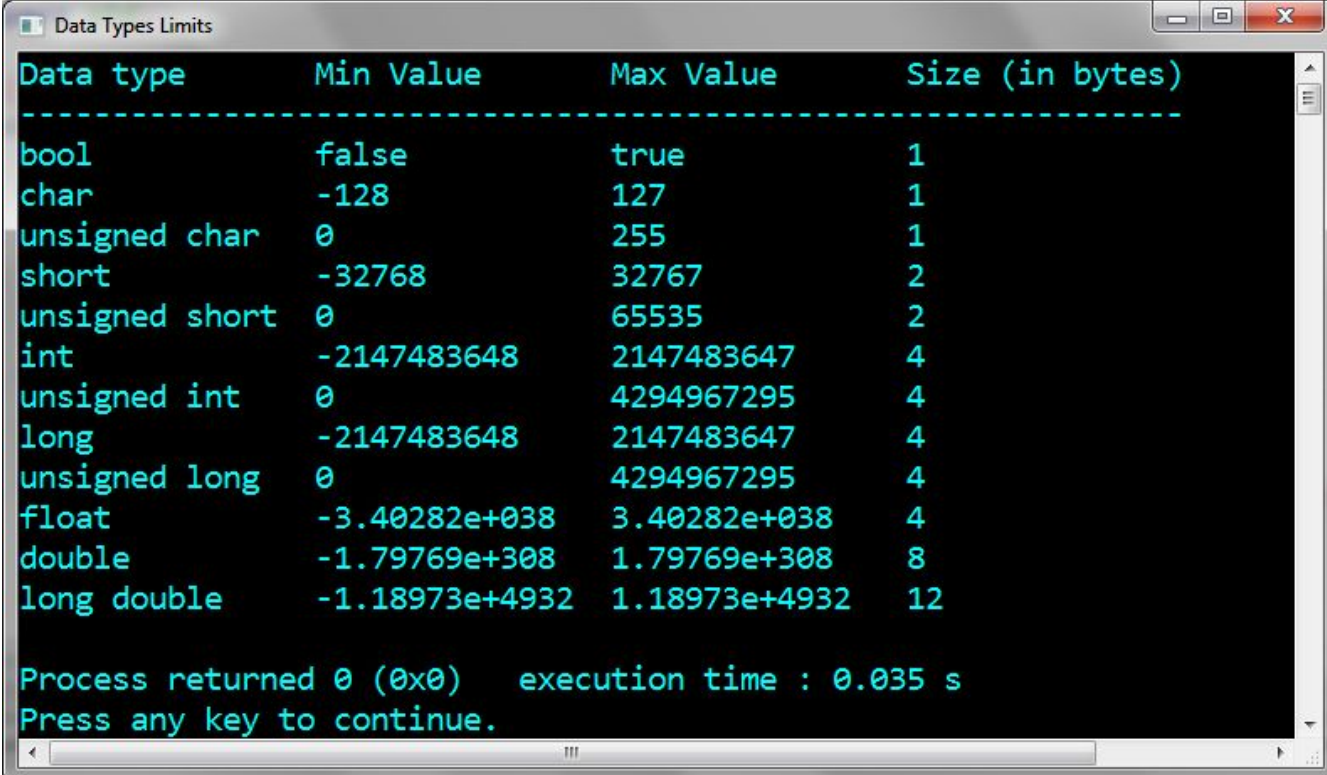
8.1. Модель данных

- ❖ Оператор `sizeof()` возвращает значение типа `size_t` — базовый беззнаковый целочисленный тип языка C++.
- ❖ Тип `size_t` представляет размер объекта в байтах.
- ❖ Размер типа выбирается таким образом, чтобы в него можно было записать максимальный размер теоретически возможного массива любого типа. Например, на 32-битной системе `size_t` будет занимать 32-бита, на 64-битной — 64-бита.
- ❖ Тип `size_t` обычно применяется для счетчиков циклов, индексации массивов, хранения размеров, адресной арифметики.
- ❖ Максимально допустимым значением типа `size_t` является значение константы `SIZE_MAX`:

```
size_t x = SIZE_MAX;  
cout << x << endl; // 4294967295
```

8. Указатели. Динамическое распределение памяти

8.1. Модель данных



```
Data Types Limits
-----
Data type      Min Value      Max Value      Size (in bytes)
-----
bool           false          true           1
char           -128           127            1
unsigned char  0              255            1
short          -32768         32767          2
unsigned short 0              65535          2
int            -2147483648    2147483647     4
unsigned int   0              4294967295     4
long           -2147483648    2147483647     4
unsigned long  0              4294967295     4
float          -3.40282e+038  3.40282e+038   4
double         -1.79769e+308  1.79769e+308   8
long double    -1.18973e+4932 1.18973e+4932  12

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

8. Указатели. Динамическое распределение памяти

8.2. Организация оперативной памяти (x86)

- ❖ **Оперативная память** — упорядоченная последовательность ячеек (байт), предназначенных для размещения данных, которыми оперирует программа во время своего выполнения.
- ❖ **Адрес байта** — порядковый номер каждого элемента последовательности (каждой ячейки памяти).
- ❖ **Адресное пространство** — непрерывный диапазон ячеек, доступный для адресации в конкретной операционной системе.

8. Указатели. Динамическое распределение памяти

8.2. Организация оперативной памяти (x86)

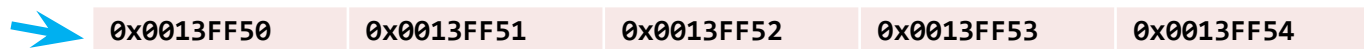
Биты



Байты (8 бит)



Адрес байта



Слово (4 байт)



Адрес слова

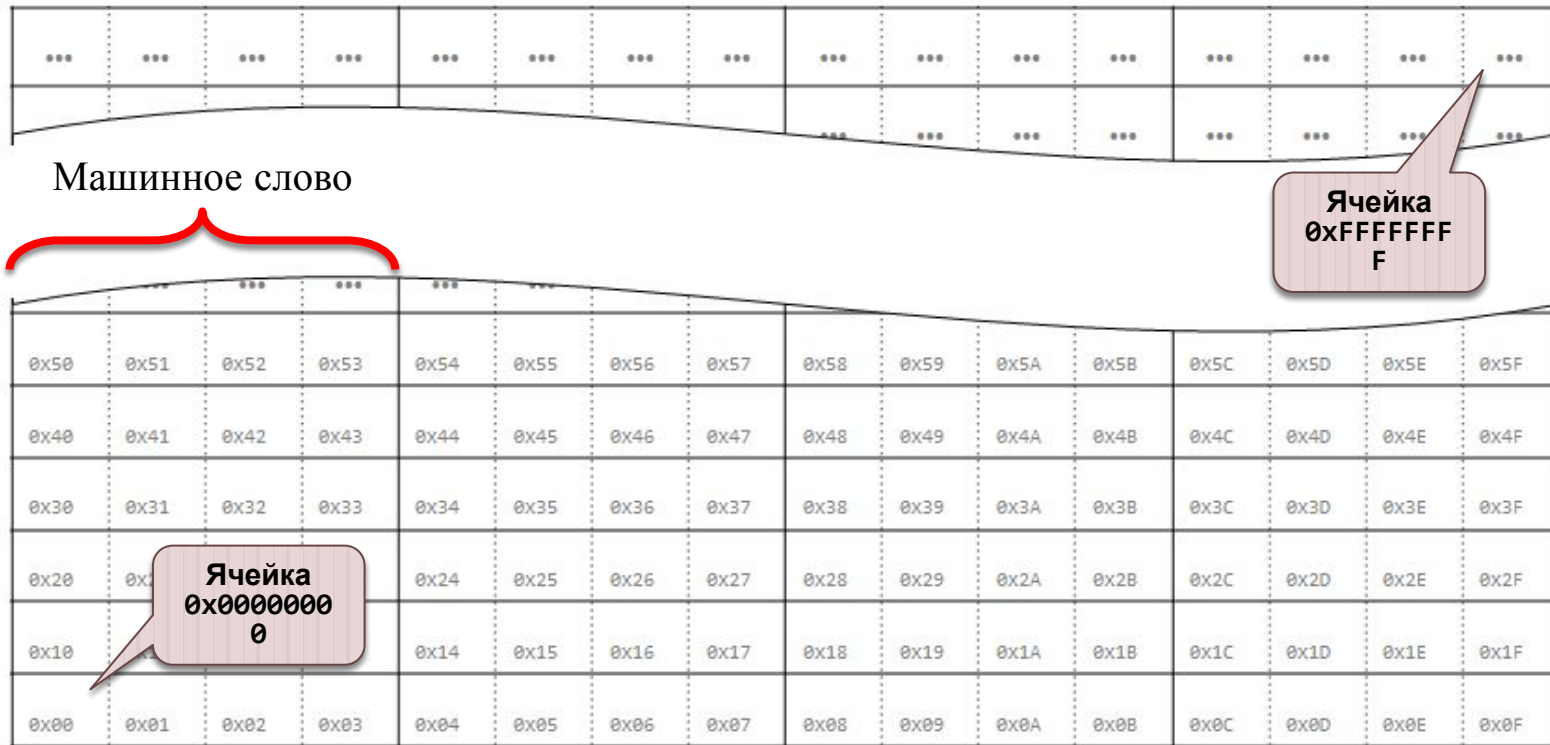


Для 32-битных систем:

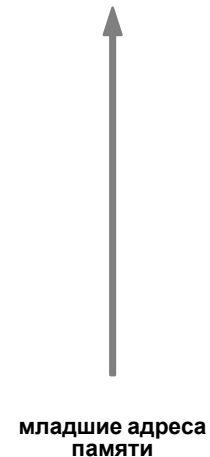
$$\begin{aligned}
 &1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 = 0\text{х}\text{FFFFFFF} = \\
 &= 4294967295 \text{ байт} = 4 \text{ Гб}
 \end{aligned}$$

8. Указатели. Динамическое распределение памяти

8.2. Организация оперативной памяти (x86)



4'294'967'295
байт
4 Гб
старшие адреса
памяти



...
...
0x50	0x51	0x52	0x53	0x54	0x55	0x56	0x57	0x58	0x59	0x5A	0x5B	0x5C	0x5D	0x5E	0x5F
0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F
0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x3A	0x3B	0x3C	0x3D	0x3E	0x3F
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27	0x28	0x29	0x2A	0x2B	0x2C	0x2D	0x2E	0x2F
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

8. Указатели. Динамическое распределение памяти

8.3. Понятие указателя

- ❖ **Указатель (*pointer*)** — это переменная, в которой хранится адрес другой переменной определенного типа.
 - Значением указателя является адрес, начиная с которого размещается в памяти переменная, на которую ссылается указатель.
 - По описанию указателя компилятор получает информацию о том, какова длина области памяти, на которую ссылается указатель (которую занимает переменная, на которую он ссылается) и о том, как интерпретировать данные в этой области памяти.
- ❖ Таким образом, переменная-указатель обладает именем и имеет тип, определяющий на какого рода данные она может ссылаться.

8. Указатели. Динамическое распределение памяти

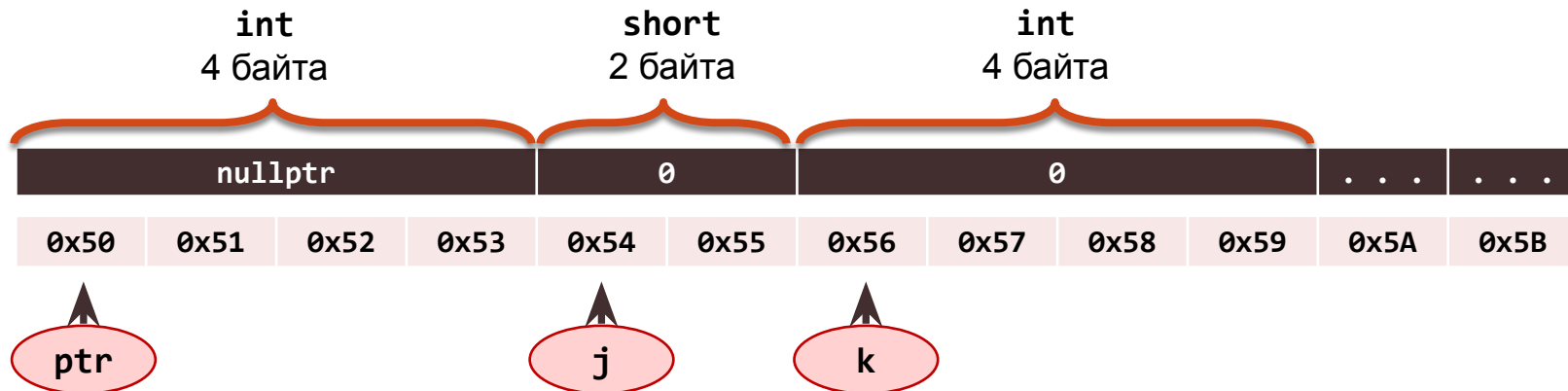
8.4. Объявление указателя

Объявление указателя: *Базовый_тип *Имя_Указателя*

```
short j = 0;
```

```
int k = 0;
```

```
int *ptr = nullptr;
```



8. Указатели. Динамическое распределение памяти

8.5. Варианты синтаксиса объявления указателя

- ❖ Первый вариант: *Базовый_тип *Имя_Указателя;*
 - Позволяет объявить несколько указателей в одной инструкции:
`int *pa, *pb, *pc;`

- ❖ Второй вариант: *Базовый_тип* Имя_Указателя;*
 - Позволяет идентифицировать тип «указатель на базовый тип» как *Базовый_тип**. Однако, эта форма записи неоднозначна при множественном объявлении переменных:
`int* pa, pb, pc;`
 - Во избежание путаницы не следует объявлять в одной инструкции более одной (указательной) переменной.

- ❖ В качестве базового типа можно указать **void**:
`void *p;`
 - Указатель на **void** не может быть разыменован!

8. Указатели. Динамическое распределение памяти

8.6. Инициализация указателя

- ❖ Указатель можно инициализировать адресом переменной, которая уже определена:

```
double dvar = 0.0;  
double *pvar = &dvar;
```

- ❖ Инициализация значением **0 (nullptr)** гарантирует, что указатель не содержит адреса, который воспринимается как корректный, а значение можно проверить в инструкции **if**:

```
int *ptr = nullptr;  
if (!ptr) cout << "ptr is null!";
```

8. Указатели. Динамическое распределение памяти

8.6. Инициализация указателя

```
7   int x = 0;
8   int *p_int_1 = 0, p_int_2 = 0;
9   double *p_dbl = 0;
10  void *p_void = 0;
11
12  p_int_1 = &x;
13  p_int_2 = p_int_2;
14  p_dbl = p_int_1;
15  p_void = p_int_1;
16
17  return 0;
18 }
```

Ошибка!
Нельзя присваивать друг другу указатели разных типов

logs & others

```
Code::Blocks x Search results x Cccc x Build log x Build messages x CppCheck x CppCheck messages x Cscor
----- Build: Release in ppointer (compiler: GNU GCC Compiler)-----
mingw32-g++.exe -Wall -fexceptions -O2 -ansi -c "D:\Учебная работа\Программирование-2014\03-Примеры\ppointer\main.cpp" -o obj\Relea
D:\Учебная работа\Программирование-2014\03-Примеры\ppointer\main.cpp: In function 'int main()':
D:\Учебная работа\Программирование-2014\03-Примеры\ppointer\main.cpp:14:13: error: cannot convert 'int*' to 'double*' in assignment
Process terminated with status 1 (0 minute(s), 0 second(s))
1 error(s), 0 warning(s) (0 minute(s), 0 second(s))
```

8. Указатели. Динамическое распределение памяти

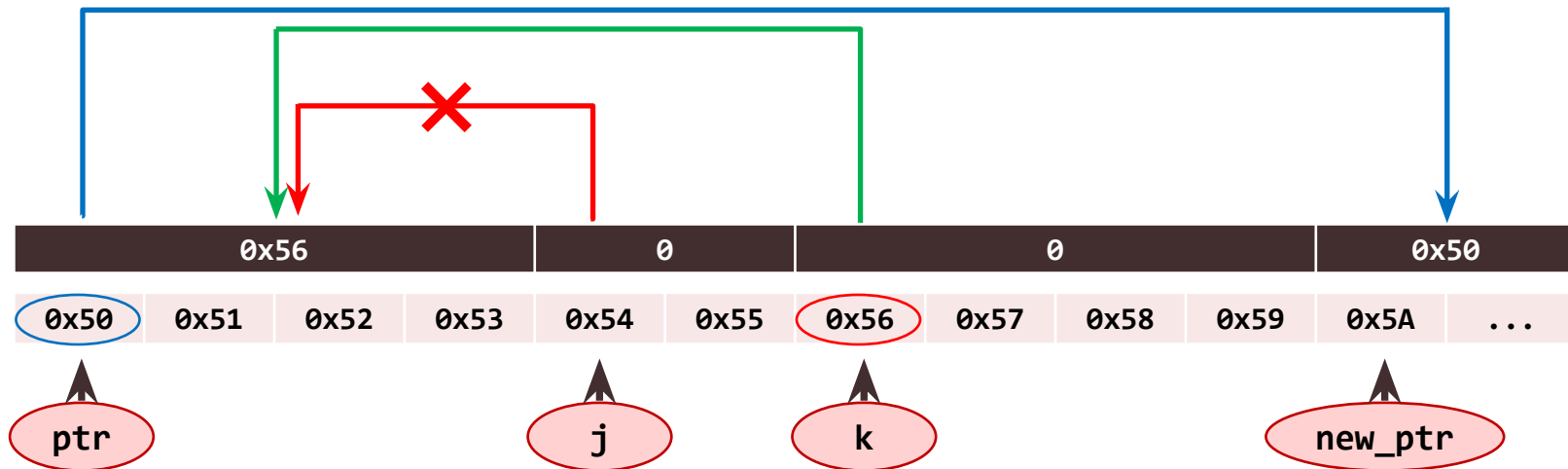
8.7. Операция взятия адреса

Операция взятия адреса: *&Имя_Переменной_или_Указателя*

```
ptr = &k;
```

~~ptr = &j;~~ — нельзя! (`ptr` – указатель на `int`, а `j` объявлена как `short`)

```
int *new_ptr = &ptr;
```



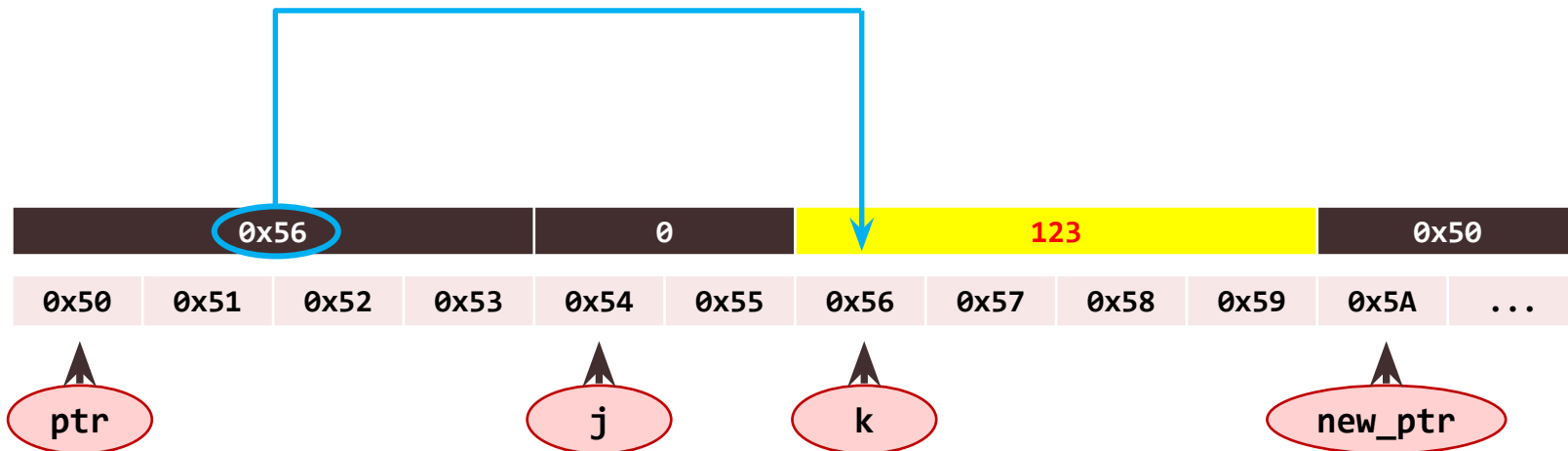
8. Указатели. Динамическое распределение памяти

8.8. Операция разыменовывания

Операция разыменовывания: **Имя_Указателя*

```
*ptr = 123;
```

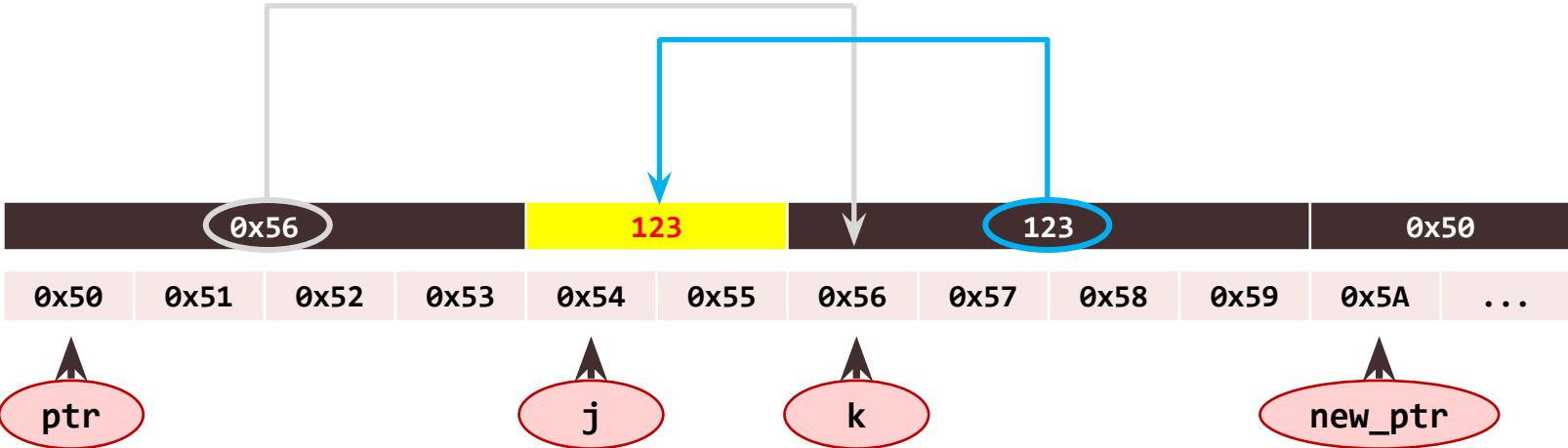
```
cout << k; // k = 123
```



8. Указатели. Динамическое распределение памяти

8.8. Операция разыменовывания

```
j = *ptr;  
cout << j; // j = k = 123
```



8. Указатели. Динамическое распределение памяти

8.9. Сравнение указателей

- ❖ Сравнить между собой имеет смысл только указатели, указывающие на объекты одного базового типа.
- ❖ Как правило, указатели сравнивают, когда они ссылаются на один и тот же объект, например, массив.

```
int x = 456;
int *ptr1 = &x;
int *ptr2 = &x;
if (ptr1 == ptr2)
    cout << "OK" << endl;
```

```
int x = 456;
int *ptr1 = &x;
double *ptr2 = &x;
if (ptr1 == ptr2)
    cout << "OK" << endl;
```


8. Указатели. Динамическое распределение памяти

8.10. Арифметика указателей

- ❖ К указателям можно применять только две арифметические операции: **сложения (добавления целого числа) и вычитания**.
- ❖ При добавлении к указателю (вычитании от указателя) целого числа n значение указателя увеличивается (уменьшается) на величину $n \times L$, где L – длина базового типа, на который ссылается указатель.
- ❖ Указатели можно вычитать. Результатом вычитания является **количество объектов базового типа**, которые можно расположить между указателями.
- ❖ **Нельзя** вычитать друг из друга указатели различных типов!
- ❖ **Нельзя** складывать указатели, даже если они имеют один и тот же тип!

Использовать арифметику указателей, как правило, имеет смысл только применительно к элементам массивов.

8. Указатели. Динамическое распределение памяти

8.11. Размещение переменных и массивов в памяти

```
int main() {  
    short c1 = '1', c2 = '2', c3 = '3';  
    short c4[3] = {1,2,3};  
  
    cout << &c1 << endl;      // 0x13FF70  
    cout << &c2 << endl;      // 0x13FF68  
    cout << &c3 << endl;      // 0x13FF6C  
    cout << endl;  
  
    cout << &c4[0] << endl;    // 0x13FF76  
    cout << &c4[1] << endl;    // 0x13FF78  
    cout << &c4[2] << endl;    // 0x13FF7A  
  
    return 0;  
}
```

12 байт между адресами переменных

2 байта между адресами элементов массива, строго упорядочены

8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

- ❖ При объявлении массива его имя является указателем на первый элемент массива. Так, если объявлен массив `k[]` и указатель `p`:

```
short k[4];
```

```
short *p;
```

то идентификатор `k` будет восприниматься как *указатель на первый элемент*, значение которого нельзя изменить (константный указатель).

```
k[0] ⇔ *k;
```

```
k[1] ⇔ *(k + 1);
```

```
/* ... */
```

- ❖ С именем массива можно работать как с указателем на первый элемент:

```
p = k;
```

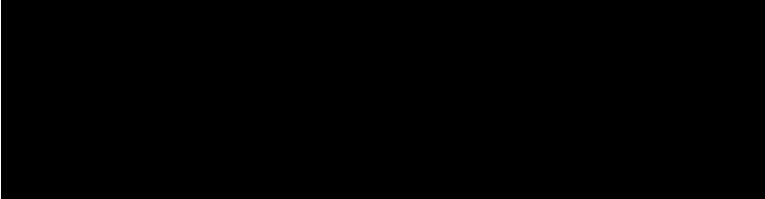
что равносильно следующему:

```
p = &k[0];
```

8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



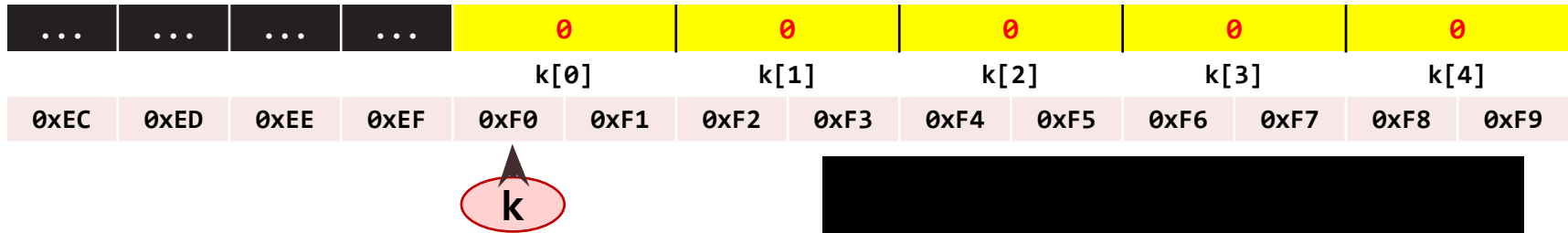
```
short k[5] = {0};
short *p = &k[0];
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i;
}
cout << &p << endl;
cout << p << " " << &k[0] << " " << k << endl;
cout << *p << " " << k[0] << " " << *k << endl;
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



```
short k[5] = {0};
```

```
short *p = &k[0];
```

```
for (int i = 0; i < 5; i++) {
```

```
    *(k+i) = i * i;
```

```
}
```

```
cout << &p << endl;
```

```
cout << p << " " << &k[0] << " " << k << endl;
```

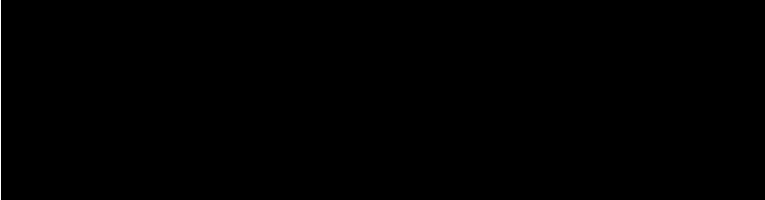
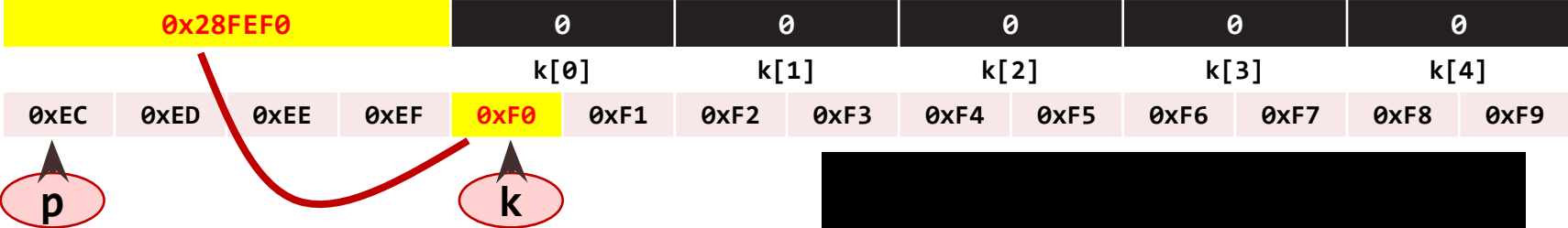
```
cout << *p << " " << k[0] << " " << *k << endl;
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



```

short k[5] = {0};
short *p = &k[0]; // Указатель на первый элемент (то же самое: *p = k)
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i;
}
cout << &p << endl;
cout << p << " " << &k[0] << " " << k << endl;
cout << *p << " " << k[0] << " " << *k << endl;

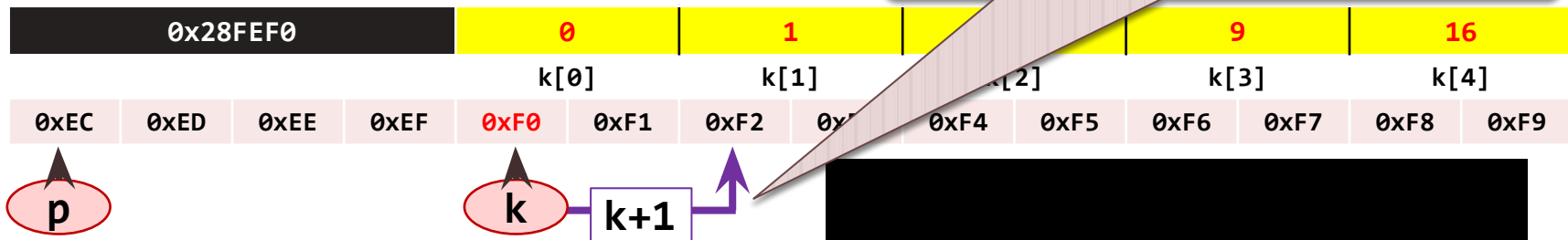
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



```

short k[5] = {0};
short *p = &k[0]; // Указатель на первый элемент (то же самое: *p = k)
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i; // k+i – переход к следующему элементу
}

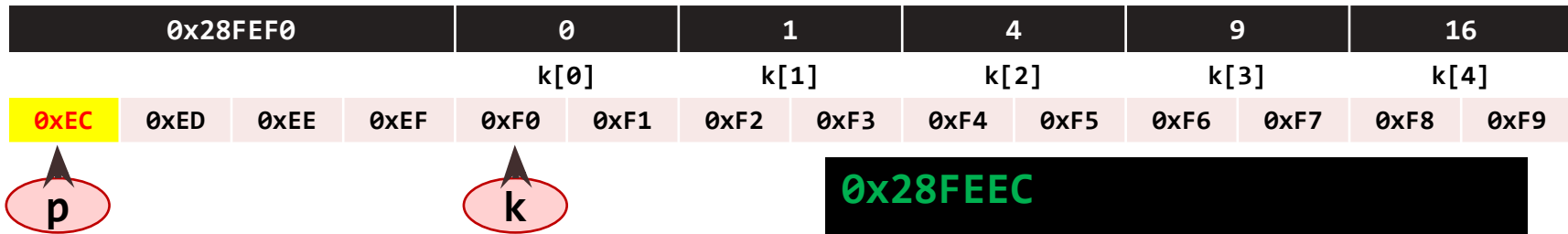
cout << &p << endl;
cout << p << " " << &k[0] << " " << k << endl;
cout << *p << " " << k[0] << " " << *k << endl;

```

8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



```

short k[5] = {0};
short *p = &k[0]; // Указатель на первый элемент (то же самое: *p = k)
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i; // k+i – переход к следующему элементу
}
cout << &p << endl; // По этому адресу расположен указатель
cout << p << " " << &k[0] << " " << k << endl;
cout << *p << " " << k[0] << " " << *k << endl;

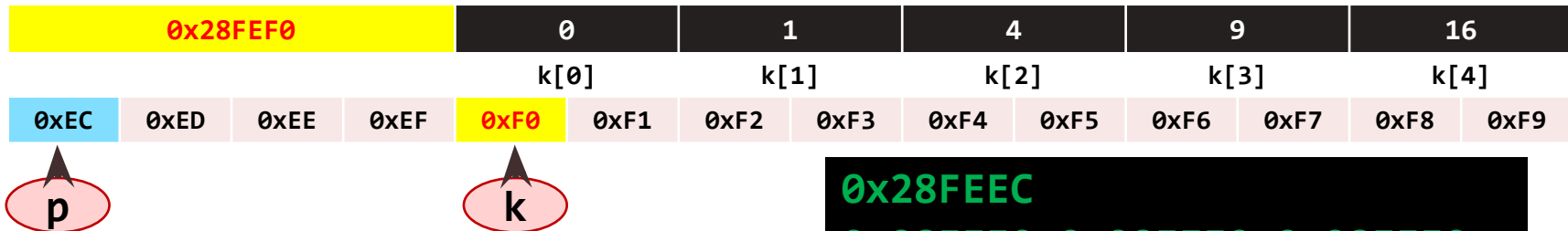
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



```
0x28FEEC
0x28FEF0 0x28FEF0 0x28FEF0
```

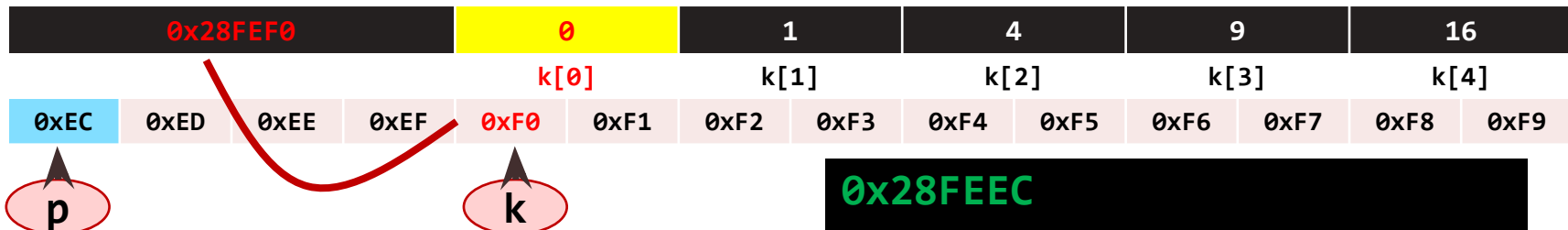
```
short k[5] = {0};
short *p = &k[0]; // Указатель на первый элемент (то же самое *p = k)
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i; // k+i – переход к следующему элементу
}
cout << &p << endl; // По этому адресу расположен указатель
cout << p << " " << &k[0] << " " << k << endl; // значение указателя
cout << *p << " " << k[0] << " " << *k << endl;
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



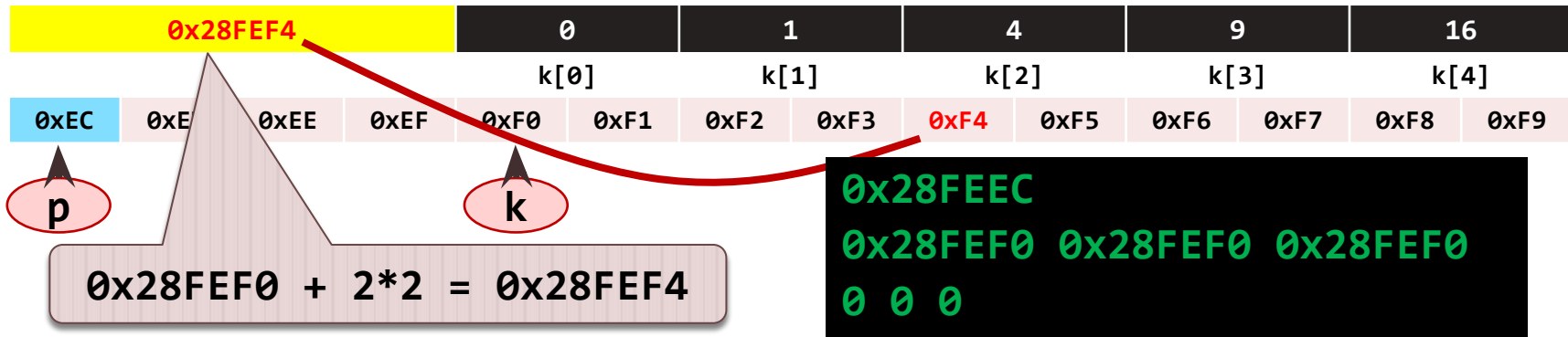
```
0x28FEEC
0x28FEF0 0x28FEF0 0x28FEF0
0 0 0
```

```
short k[5] = {0};
short *p = &k[0]; // Указатель на первый элемент (то же самое *p = k)
for (int i = 0; i < 5; i++) {
    *(k+i) = i * i; // k+i – переход к следующему элементу
}
cout << &p << endl; // По этому адресу расположен указатель
cout << p << " " << &k[0] << " " << k << endl; // значение указателя
cout << *p << " " << k[0] << " " << *k << endl; // разыменованное
```

8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



`p += 2;` // `k += 2;` операция запрещена: невозможно изменить адрес массива

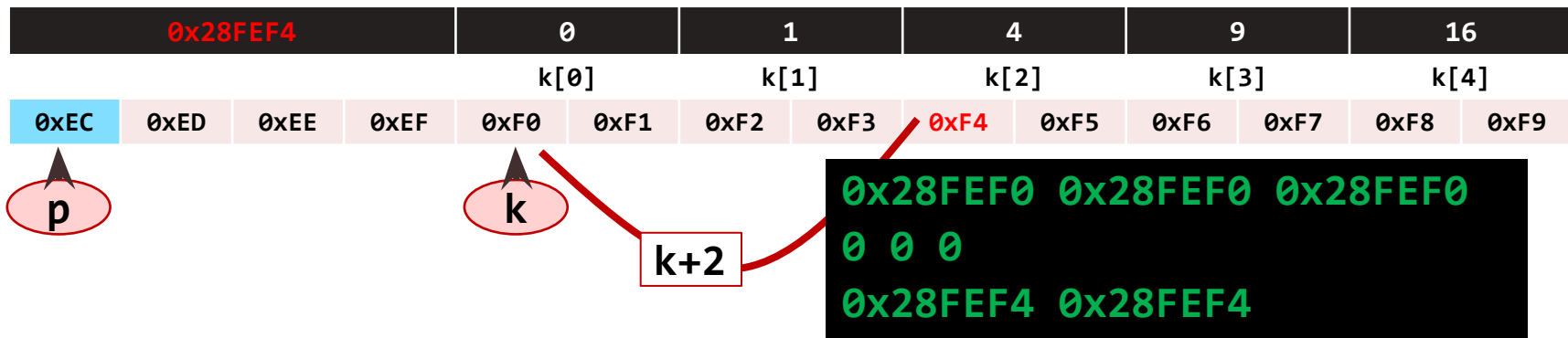
```
cout << k + 2 << " " << p << endl;
cout << *(k + 2) << " " << *p << endl;
cout << &k[4] - &k[3];
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



`p += 2;` // `k += 2;` операция запрещена: невозможно изменить адрес массива

```
cout << k + 2 << " " << p << endl;
```

```
cout << *(k + 2) << " " << *p << endl;
```

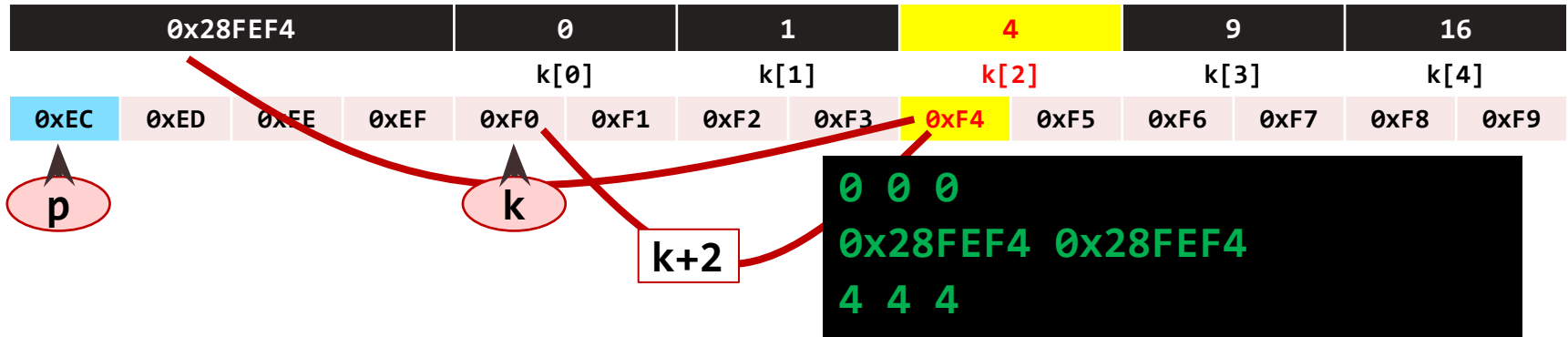
```
cout << &k[4] - &k[3];
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



`p += 2;` // `k += 2;` операция запрещена: невозможно изменить адрес массива

```
cout << k + 2 << " " << p << endl;
```

```
cout << *(k + 2) << " " << *p << " " << k[2] << endl;
```

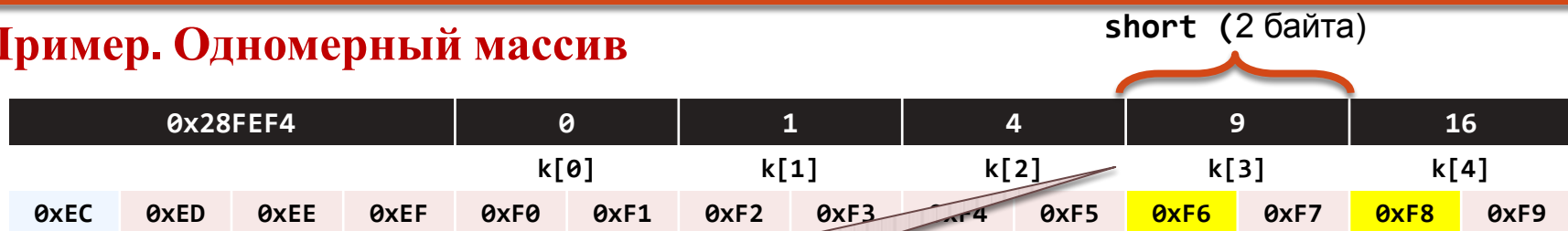
```
cout << &k[4] - &k[3];
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

Пример. Одномерный массив



$$(0x28FEF8 - 0x28FEF6) / 2 = 1$$

```
0x28FEF4 0x28FEF4
4 4 4
1
```

`p += 2; // k += 2;` операция запрещена: невозможно изменить адрес массива

```
cout << k + 2 << " " << p << endl;
```

```
cout << *(k + 2) << " " << *p << " " << k[2] << endl;
```

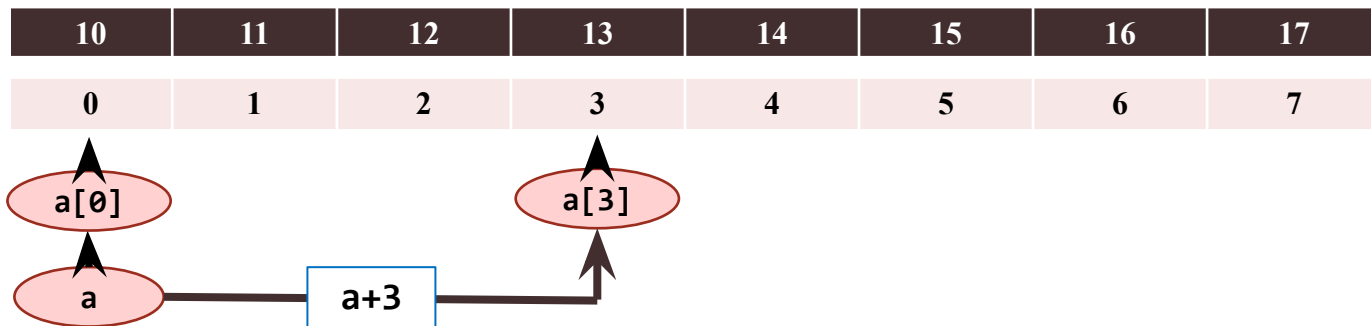
```
cout << &k[4] - &k[3];
```

8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

- ❖ Операция `[n]` применительно к идентификатору одномерного массива производит смещение на `n` элементов относительно начала и разыменовывает полученный указатель на `n`-й элемент массива.
- ❖ То же можно сделать явно, используя адресную арифметику.

```
short  a[8] = {10,11,12,13,14,15,16,17};  
cout << *a      << a[0]; // 10  
cout << *(a+3) << a[3]; // 13
```



8. Указатели. Динамическое распределение памяти

8.12. Связь указателей и массивов

- ❖ Идентификатор многомерного массива трактуется как указатель на массив (для двумерного массива — указатель на первую строку). Исходя из этого, можно поступить и иначе:

```
const int n=3, m=4;
int ai[n][m]={0,1,2,3}, {10,11,12,13}, {20,21,22,23}};
int i=2, j=2;
cout << ai[i][j] << endl; // 22
cout << (*(ai+i)+j) << endl; // 22
```



8. Указатели. Динамическое распределение памяти

8.13. Массивы указателей

- ❖ При необходимости можно описать не просто указатель на `int`, но и массив указателей на `int`:

```
int *pparint[2];
```

```
int a = 0;
```

```
int b = 1;
```

```
pparint[0] = &a;
```

```
pparint[1] = &b;
```

```
*pparint[0] = *pparint[1];
```

```
cout << pparint[0] << pparint[1]; // 0x28FEFC 0x28FEF8
```

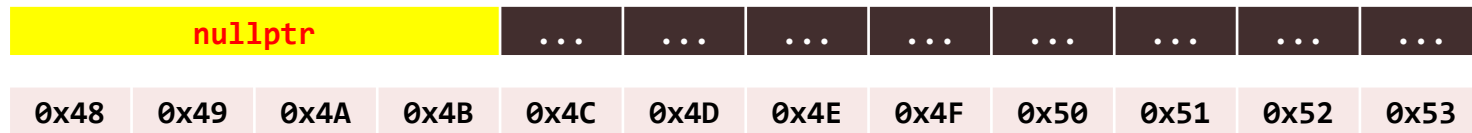
```
cout << *pparint[1] << *pparint[2]; // 1 1
```

Различные адреса
Одинаковые значения

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;
```

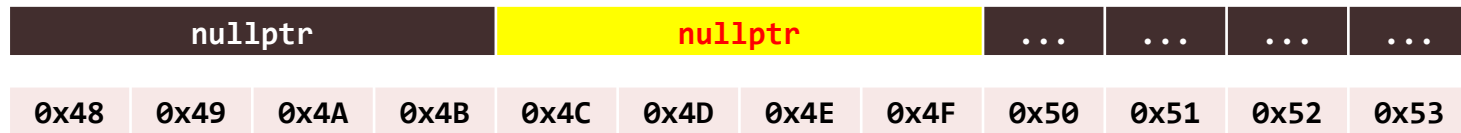


ppshort

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;
```



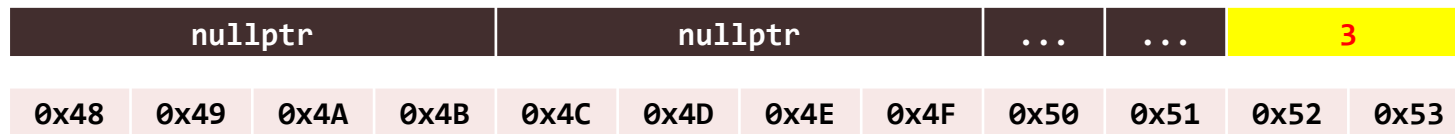
ppshort

pshort

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;
```



ppshort

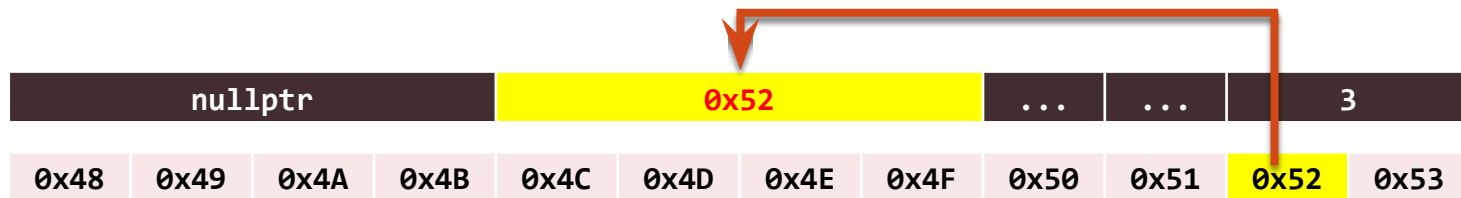
pshort

a

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;  
pshort = &a;
```



ppshort

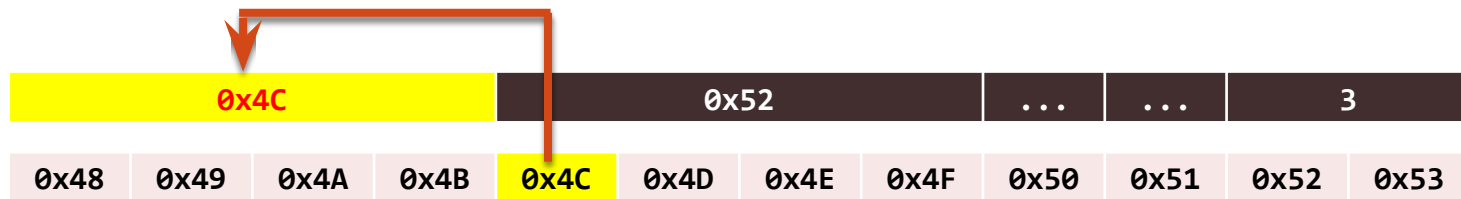
pshort

a

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;  
pshort = &a;  
ppshort = &pshort;
```



ppshort

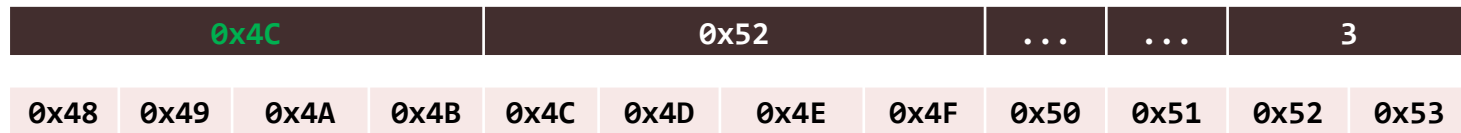
pshort

a

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;  
pshort = &a;  
ppshort = &pshort;  
cout << pshort << endl;    // 0x4C
```



ppshort

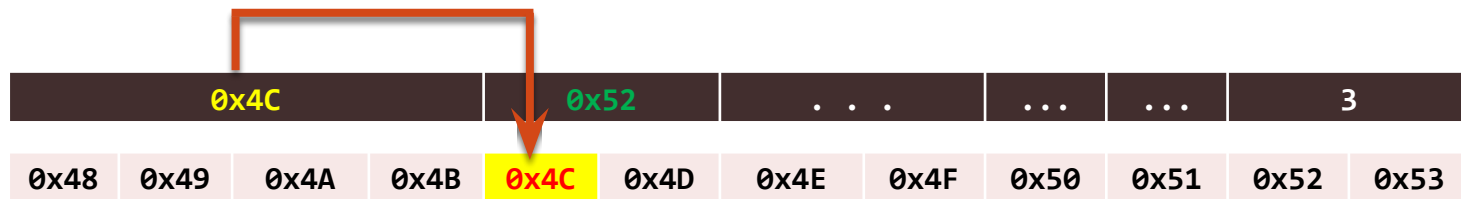
pshort

a

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;  
pshort = &a;  
ppshort = &pshort;  
cout << pshort << endl; // 0x4C  
cout << *ppshort << endl; // 0x52
```



ppshort

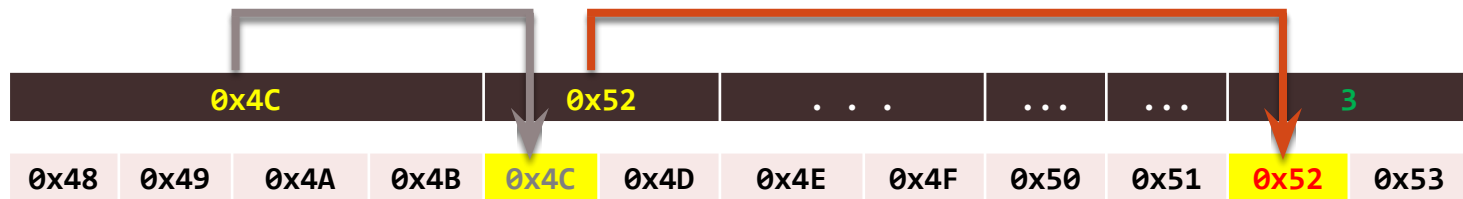
pshort

a

8. Указатели. Динамическое распределение памяти

8.14. Указатели на указатели

```
short **ppshort = nullptr;  
short *pshort = nullptr;  
short a = 3;  
pshort = &a;  
ppshort = &pshort;  
cout << pshort << endl; // 0x4C  
cout << *pshort << endl; // 0x52  
cout << **ppshort << endl; // 3
```



ppshort

pshort

a

8. Указатели. Динамическое распределение памяти

8.15. Динамическое управление памятью

- ❖ Память для глобальных переменных распределяется на этапе компиляции и выделяется при загрузке программы в память.
- ❖ Память для локальных переменных выделяется в области стека в момент начала выполнения функции.
- ❖ Невозможно в процессе выполнения программы объявить новые локальные или глобальные переменные, хотя необходимость такого объявления может возникнуть.
- ❖ Невозможно работать со структурами данных, размер которых может изменяться в процессе выполнения программы (к таким структурам относятся, например, списки, стеки, очереди, деревья).

8. Указатели. Динамическое распределение памяти

8.15. Динамическое управление памятью

- ❖ Обычно динамическое распределение памяти используется в следующих случаях:
 - когда неизвестно необходимое количество объектов;
 - когда точно неизвестно, какого типа объекты нужны;
 - когда нельзя разрешать совместное использование данных несколькими объектами.

Механизм динамического распределения памяти позволяет программе получать необходимую для хранения данных память в процессе своего выполнения.

8. Указатели. Динамическое распределение памяти

8.16. Выделение и освобождение памяти

❖ Для получения доступа к динамически выделяемым областям памяти и их типизации используются указатели.

❖ Выделение памяти:

Имя_Указателя = new Тип;

Имя_Указателя = new Тип (инициализирующее_значение);

❖ Освобождение памяти:

delete Имя_Указателя;

❖ Перед использованием оператора **new** следует объявить используемый указатель.

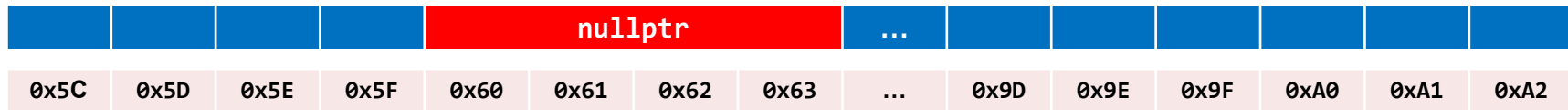
❖ Операции объявления указателя и выделения памяти можно совместить:

*Тип *Имя_Указателя = new Тип;*

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;
```

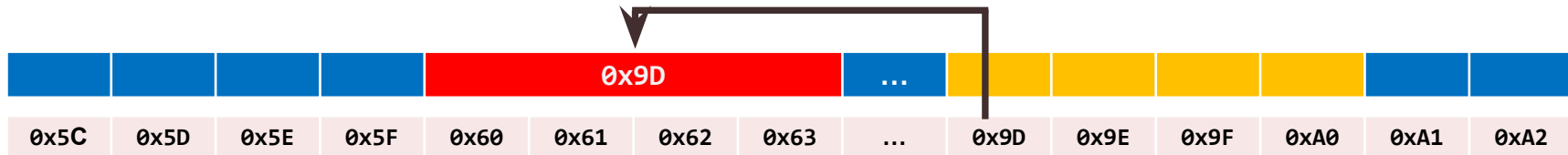


pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;  
pf = new float;
```

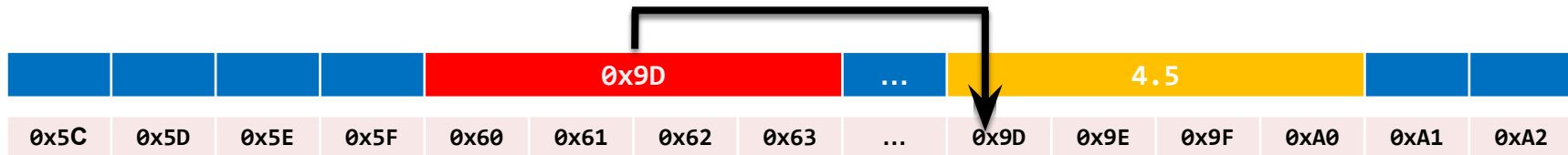


pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;
```

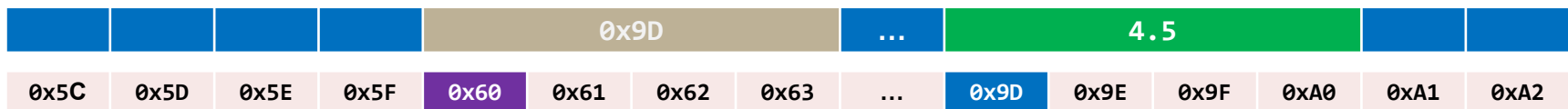


pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;  
cout << &pf << pf << *pf; // 0x60 0x9D 4.5
```

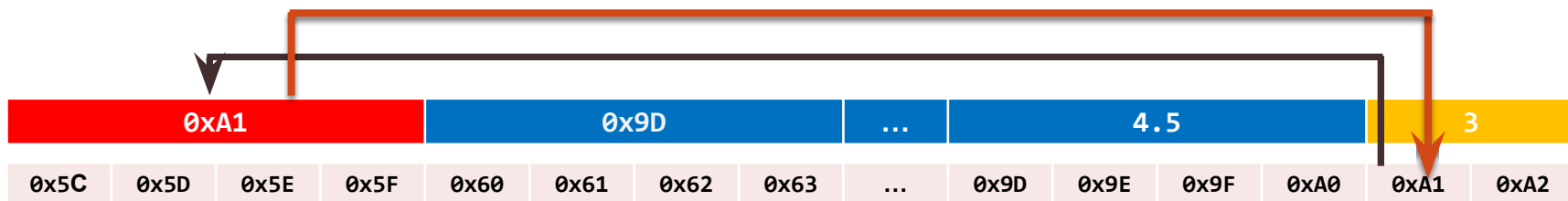


pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;  
cout << &pf << pf << *pf; // 0x60 0x9D 4.5  
short *ps = new short (3);
```



ps

pf

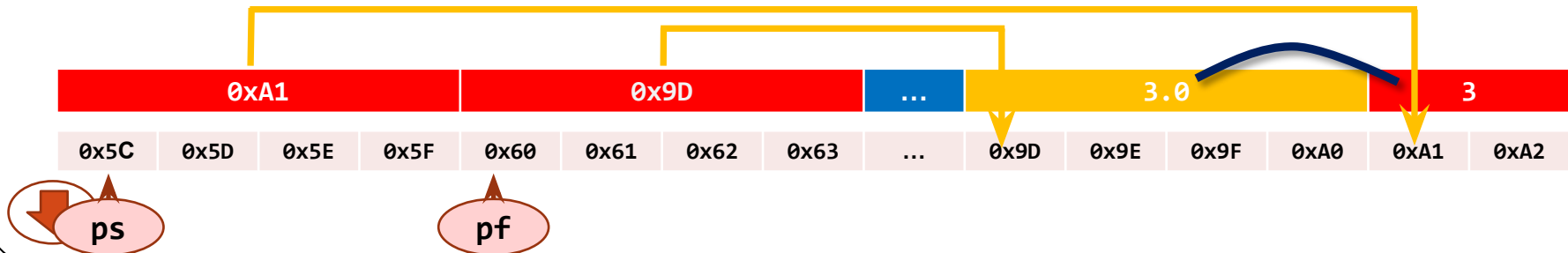
8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```

float *pf = nullptr;
pf = new float;
*pf = 4.5;
cout << &pf << pf << *pf; // 0x60 0x9D 4.5
short *ps = new short (3);
*pf = *ps;

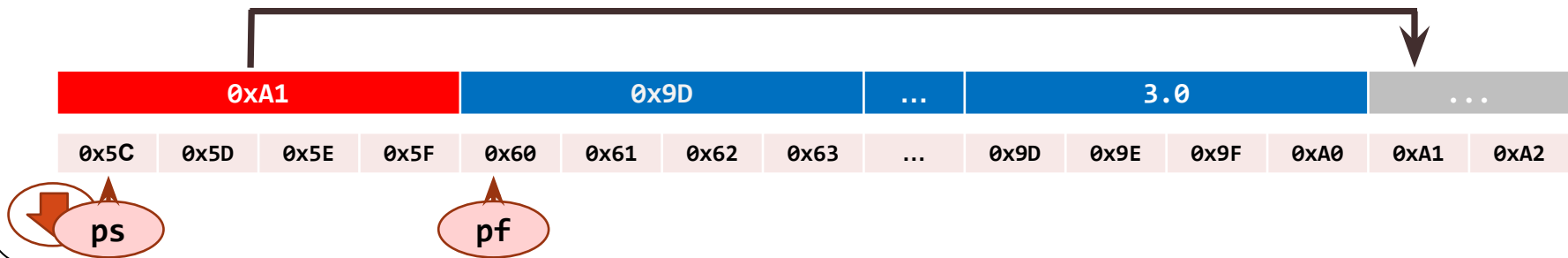
```



8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

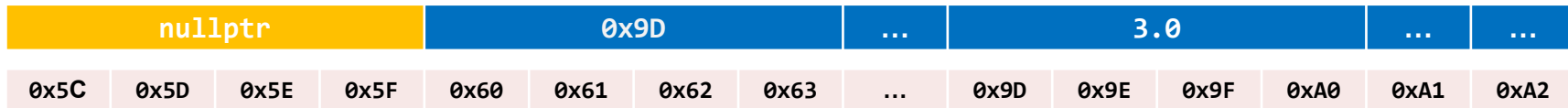
```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;  
cout << &pf << pf << *pf; // 0x60 0x9D 4.5  
short *ps = new short (3);  
*pf = *ps;  
delete ps;
```



8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;
pf = new float;
*pf = 4.5;
cout << &pf << pf << *pf; // 0x60 0x9D 4.5
short *ps = new short (3);
*pf = *ps;
delete ps;
ps = nullptr;
```



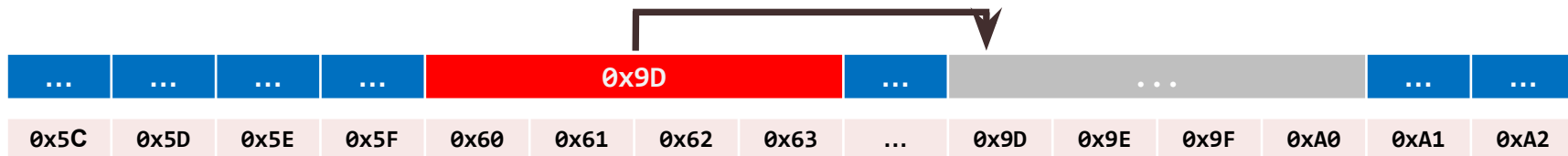
ps

pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для переменных

```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;  
cout << &pf << pf << *pf; // 0x60 0x9D 4.5  
short *ps = new short (3);  
*pf = *ps;  
delete ps;  
ps = nullptr;  
delete pf;
```



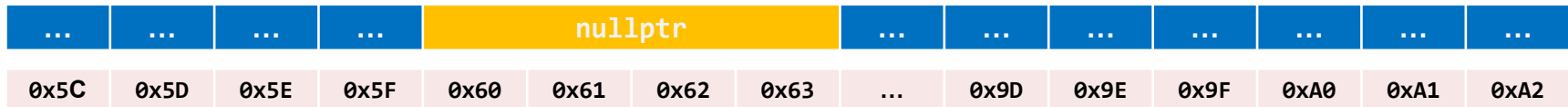
pf

8. Указатели. Динамическое распределение памяти

8.17. Динамическое выделение памяти для

переменных

```
float *pf = nullptr;  
pf = new float;  
*pf = 4.5;  
cout << &pf << pf << *pf; // 0x60 0x9D 4.5  
short *ps = new short (3);  
*pf = *ps;  
delete ps;  
ps = nullptr;  
delete pf;  
pf = nullptr;
```



8. Указатели. Динамическое распределение памяти

8.18. Динамическое выделение памяти (1D-массив)

```
int k = 0;  
cin >> k;  
int *pa = new int [k];
```

Особенности выделения памяти для массива:

- объявляется указатель на базовый тип массива;
- размер массива указывается в квадратных скобках после идентификатора типа в операторе `new`;
- размер массива — не константа;



8. Указатели. Динамическое распределение памяти

8.18. Динамическое выделение памяти (1D-массив)

```
int k = 0;
cin >> k;
int *pa = new int [k];
cout << &pa << endl;
cout << pa << endl;
for (int i = 0; i < 4; i++)
    *(pa+i) = i*i;
```

Особенности выделения памяти для массива:

- для получения доступа к элементу массива разыменовывается соответствующий указатель;



8. Указатели. Динамическое распределение памяти

8.18. Динамическое выделение памяти (1D-массив)

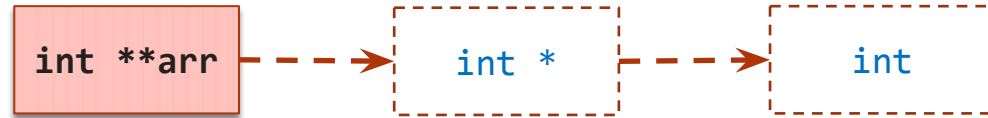
```
int k = 0;
cin >> k;
int *pa = new int [k];
cout << &pa << endl;
cout << pa << endl;
for (int i = 0; i < 4; i++)
    *(pa+i) = i*i;
cout << *pa << endl; // 0
cout << *(pa+2) << endl; // 4
delete [] pa;
```

Особенности освобождения выделенной памяти от массива:

- обязательно наличие квадратных скобок [] после оператора delete;
- размер массива в скобках не указывается.

8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)



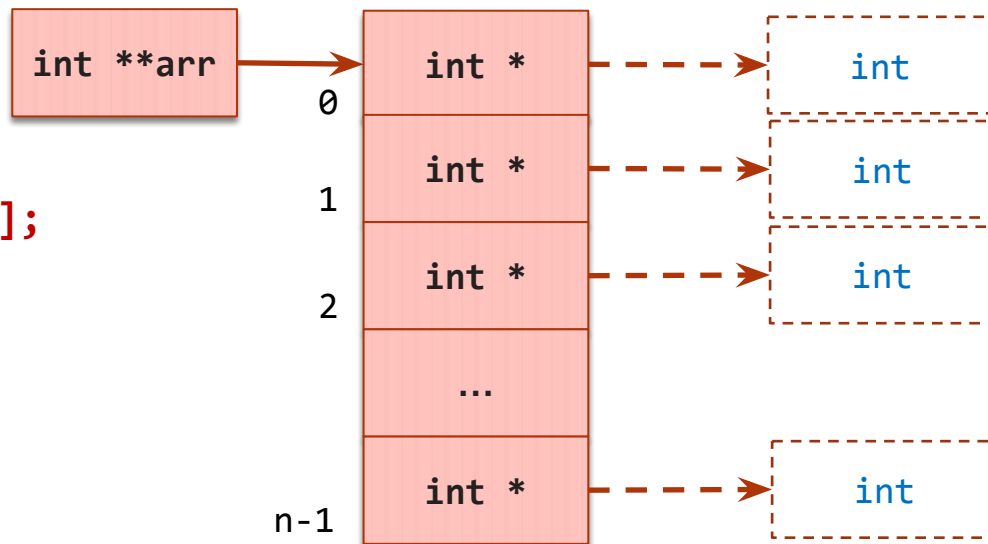
```
int **arr;
```



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

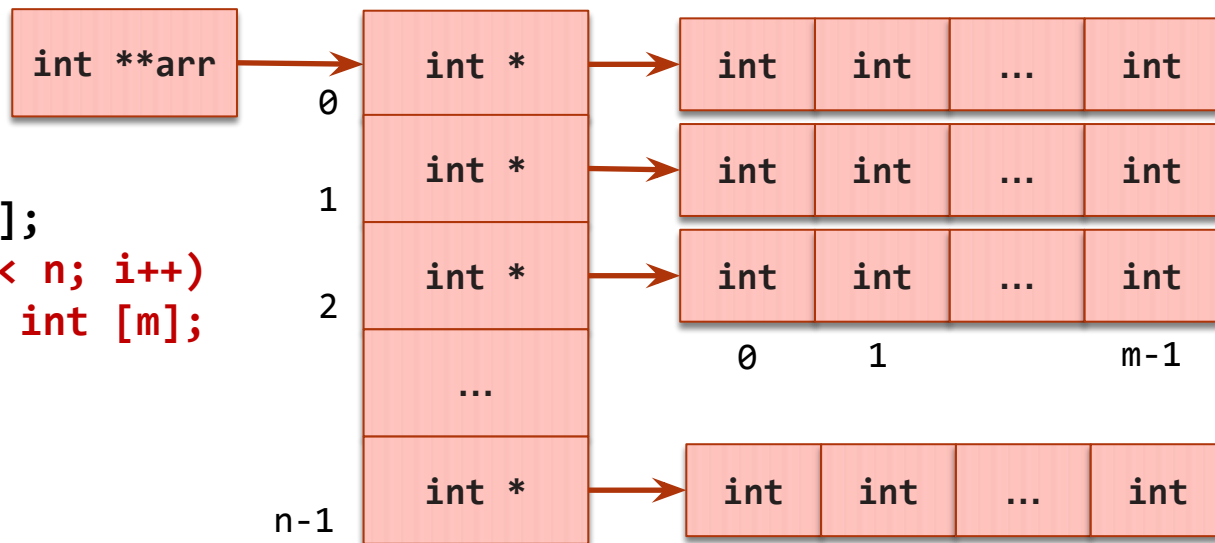
```
int **arr;  
arr = new int * [n];
```



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

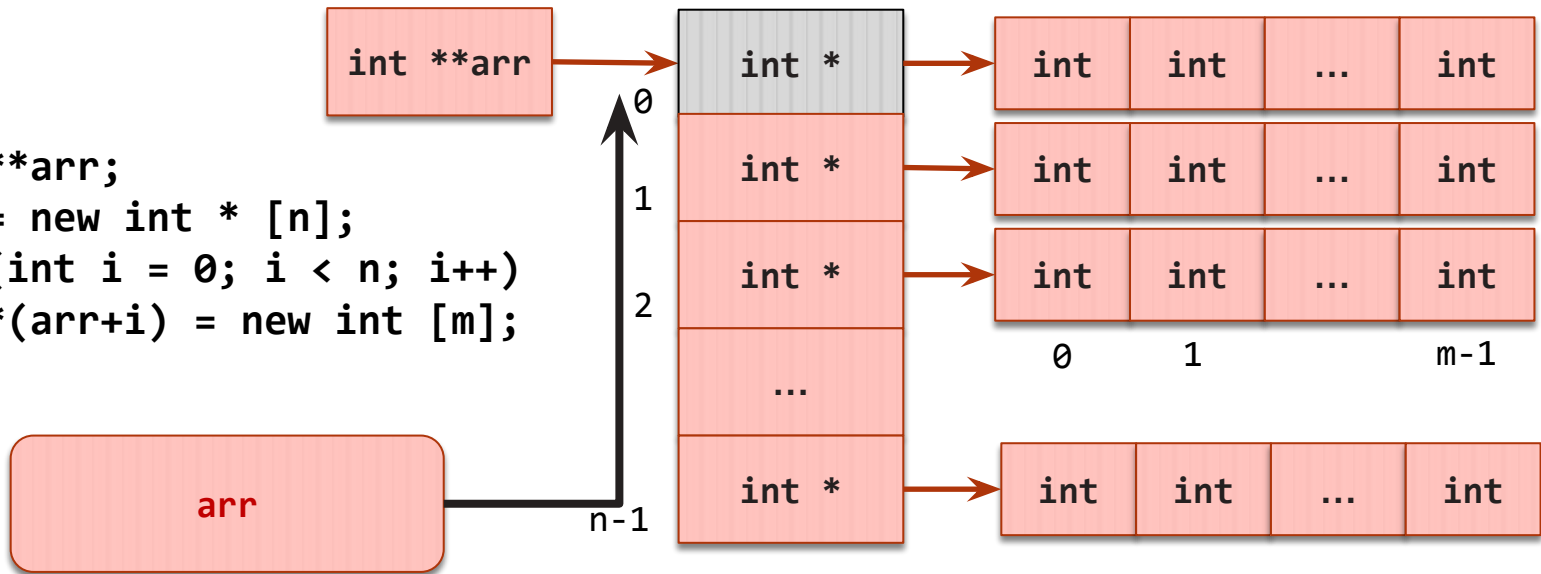
```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

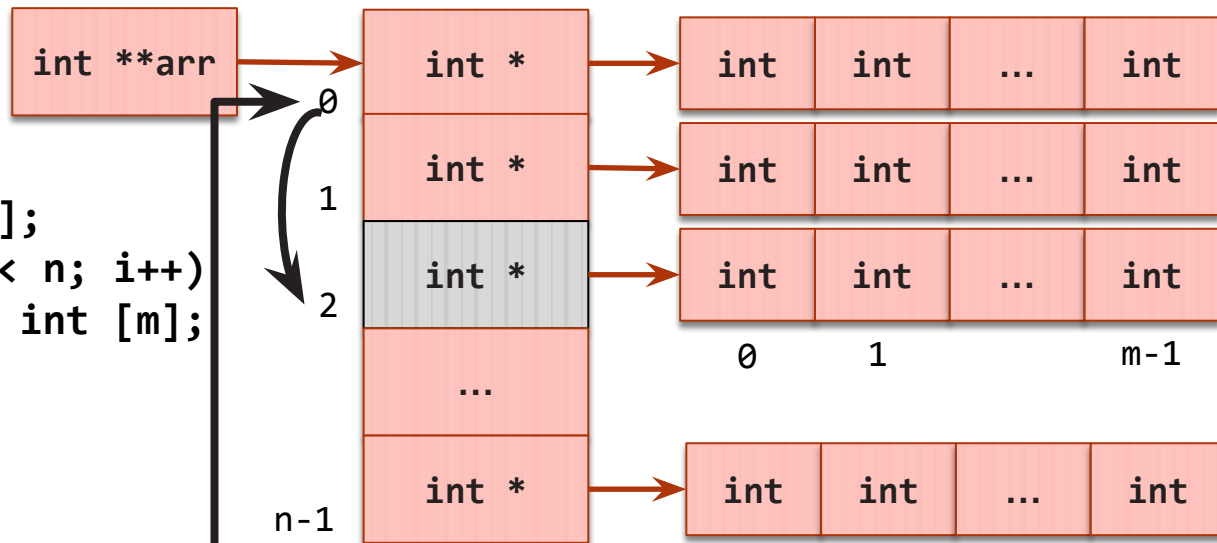
```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```



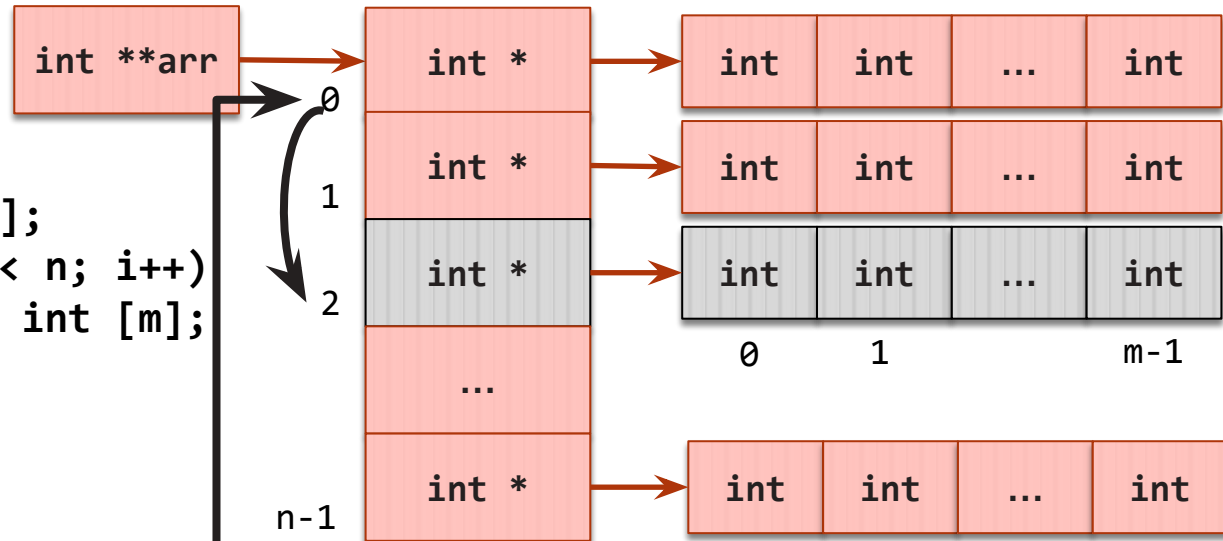
`arr + 2`



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```



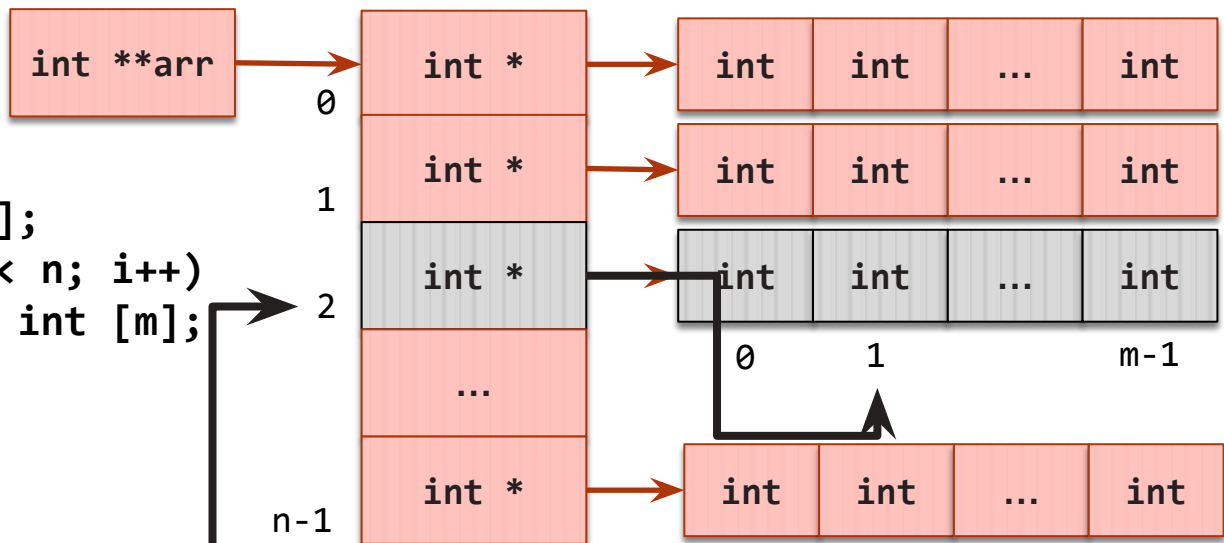
`*(arr+2)`



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```



`*(arr+2)+1`

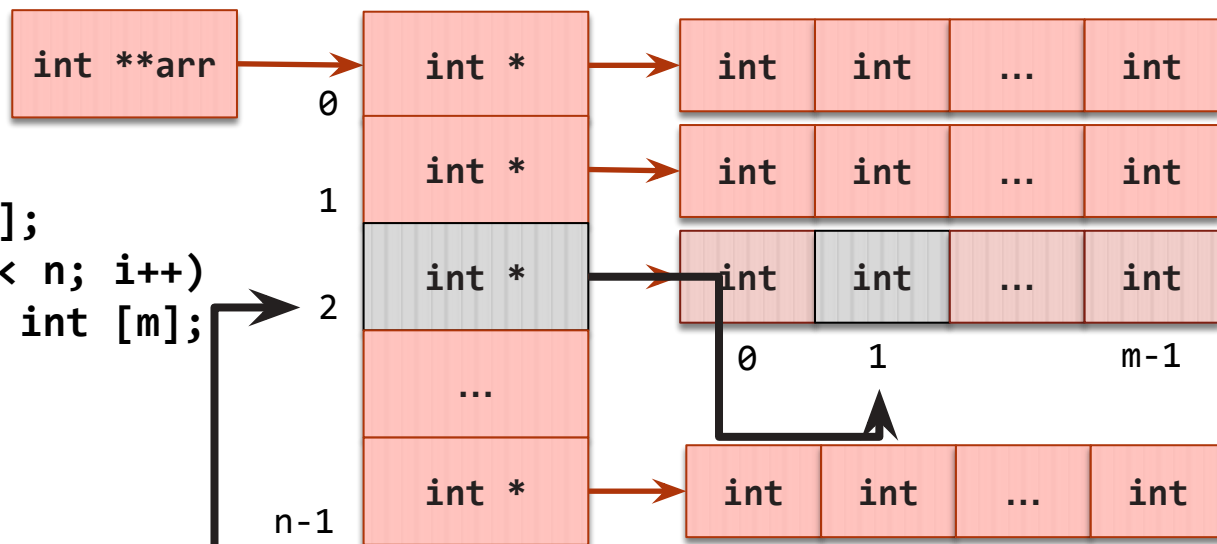


8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)

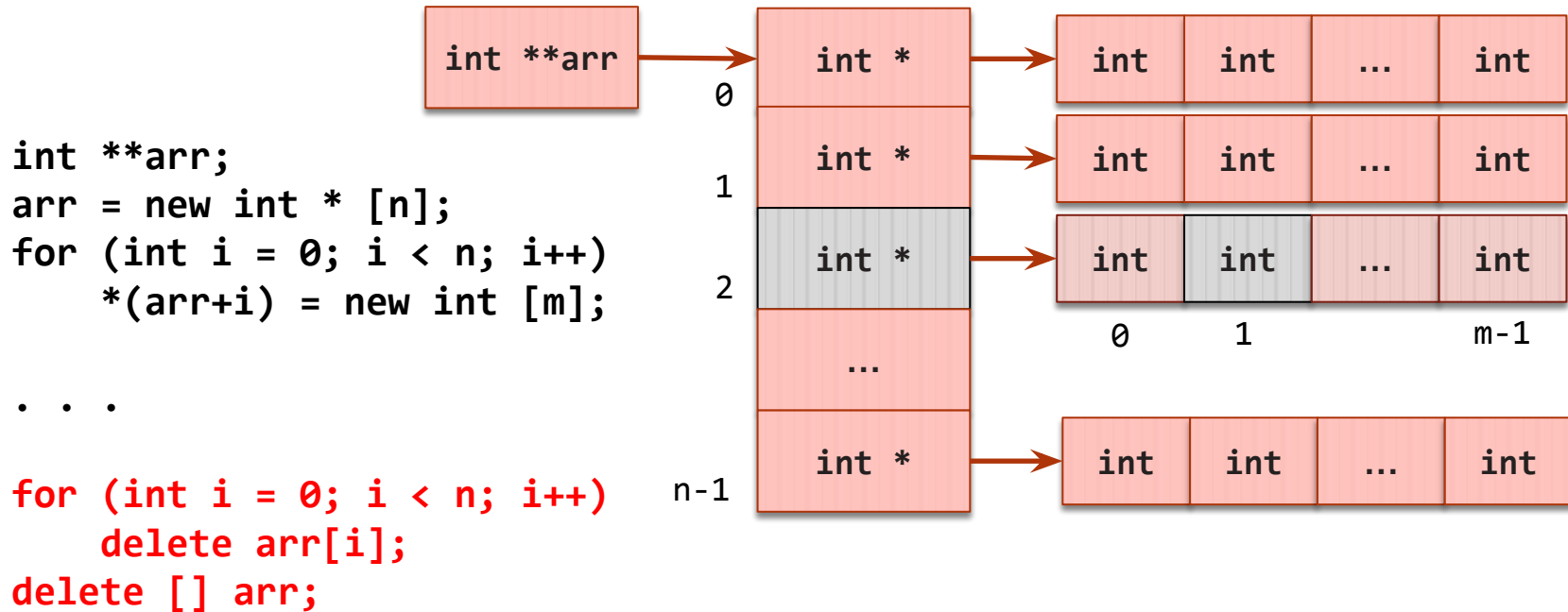
```
int **arr;  
arr = new int * [n];  
for (int i = 0; i < n; i++)  
    *(arr+i) = new int [m];
```

`*(*(arr+2)+1)`



8. Указатели. Динамическое распределение памяти

8.19. Динамическое выделение памяти (2D-массив)



8. Указатели. Динамическое распределение памяти

8.20. Ссылки

- ❖ **Ссылка** (*reference*) — псевдоним для другой переменной.
- ❖ Ссылка имеет имя, которое может использоваться вместо имени переменной.
- ❖ Так как ссылка — это псевдоним, а не указатель, переменная, для которой она определяется, должна быть объявлена ранее.
- ❖ Ссылка не может быть изменена, чтобы представлять другую переменную.
- ❖ Ссылки чаще всего используют для передачи аргументов в функции, однако иногда их можно использовать в качестве синонимов для упрощения текста программы.

8. Указатели. Динамическое распределение памяти

8.20. Ссылки

- ❖ Объявление и инициализация:

Базовый_тип &Имя_Ссылки = Имя_Переменной;

```
int number = 0;
```

```
int &rnumber = number; // ссылка на переменную
```

```
int *pnumber = &number; // указатель на адрес переменной
```

- ❖ Ссылку, можно использовать вместо имени исходной переменной:

```
rnumber += 10;
```

```
cout << rnumber << number; // 10 10
```

```
*pnumber += 10; // !!! требуется разыменованье
```

8. Указатели. Динамическое распределение памяти

**СПАСИБО ЗА
ВНИМАНИЕ!**