ЛЕКЦИЯ 08
ПРОСТЫЕ ЧИСЛА

АЛГОРИТМИЗАЦИЯ И
ПРОГРАММИРОВАНИЕ

ПРОСТЫЕ ЧИСЛА

Определение

$unit\ x \Leftrightarrow \exists u, ux = 1$

$prime\ p \Leftrightarrow \nexists x, x \neq u \cap x \neq pu \cap x \backslash \mathrm{p}$

Мнемонической правило «Делится только на самого себя и на единицу».

Но строго говоря в целых числах units это -1 и 1

Значит реально еще на -1 и на минус самого себя

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

НАИВНЫЙ АЛГОРИТМ

Это ведет к очевидному решению:

```
bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
    return true;
}
```

N-ОЕ ПРОСТОЕ ЧИСЛО

Первым простым числом является 2, а шестым – 13.

Используя алгоритм с предыдущего слайда найдите N-ое простое число

РЕШЕТО ЭРАТОСФЕНА

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

РЕШЕТО ЭРАТОСФЕНА

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

РЕШЕТО ЭРАТОСФЕНА

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

РЕШЕТО ЭРАТОСФЕНА

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

РЕШЕТО ЭРАТОСФЕНА

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

ПОСМОТРИМ НА ПАМЯТЬ

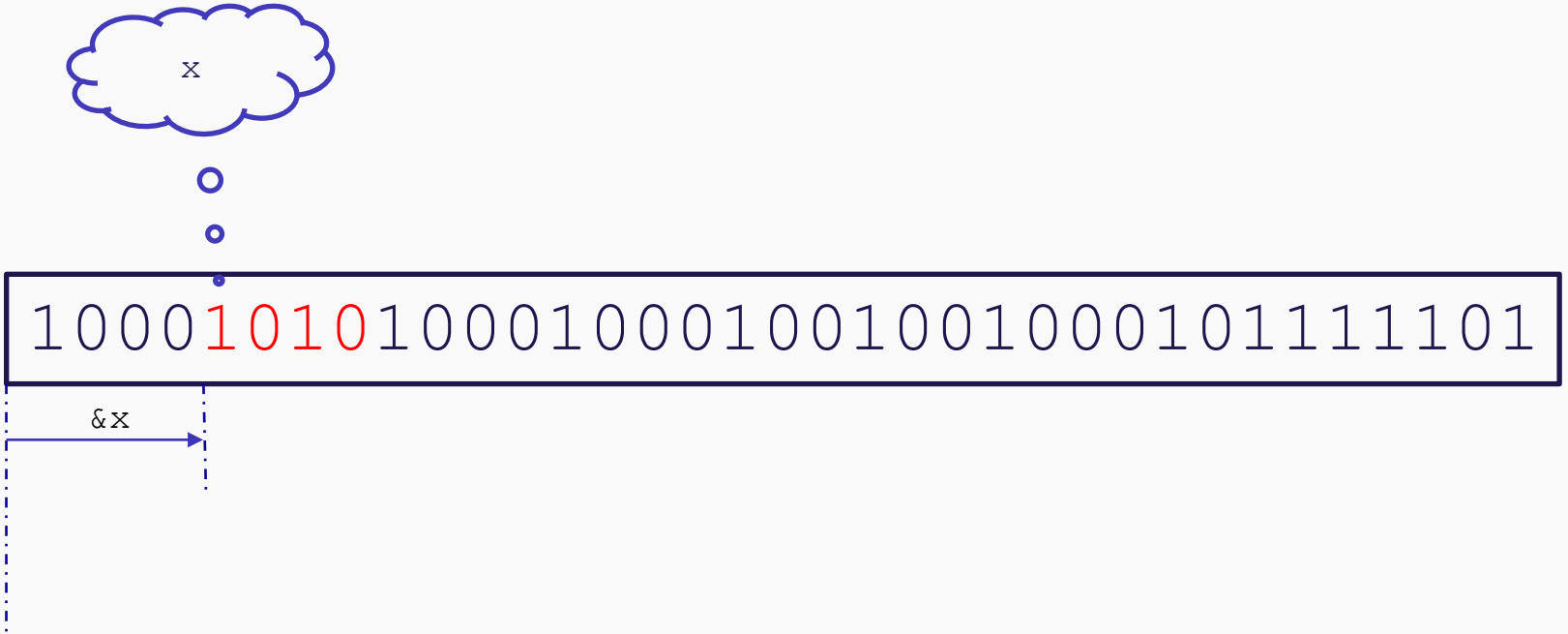
```
10001010100010001001001000101111101
```

ПОСМОТРИМ НА ПАМЯТЬ

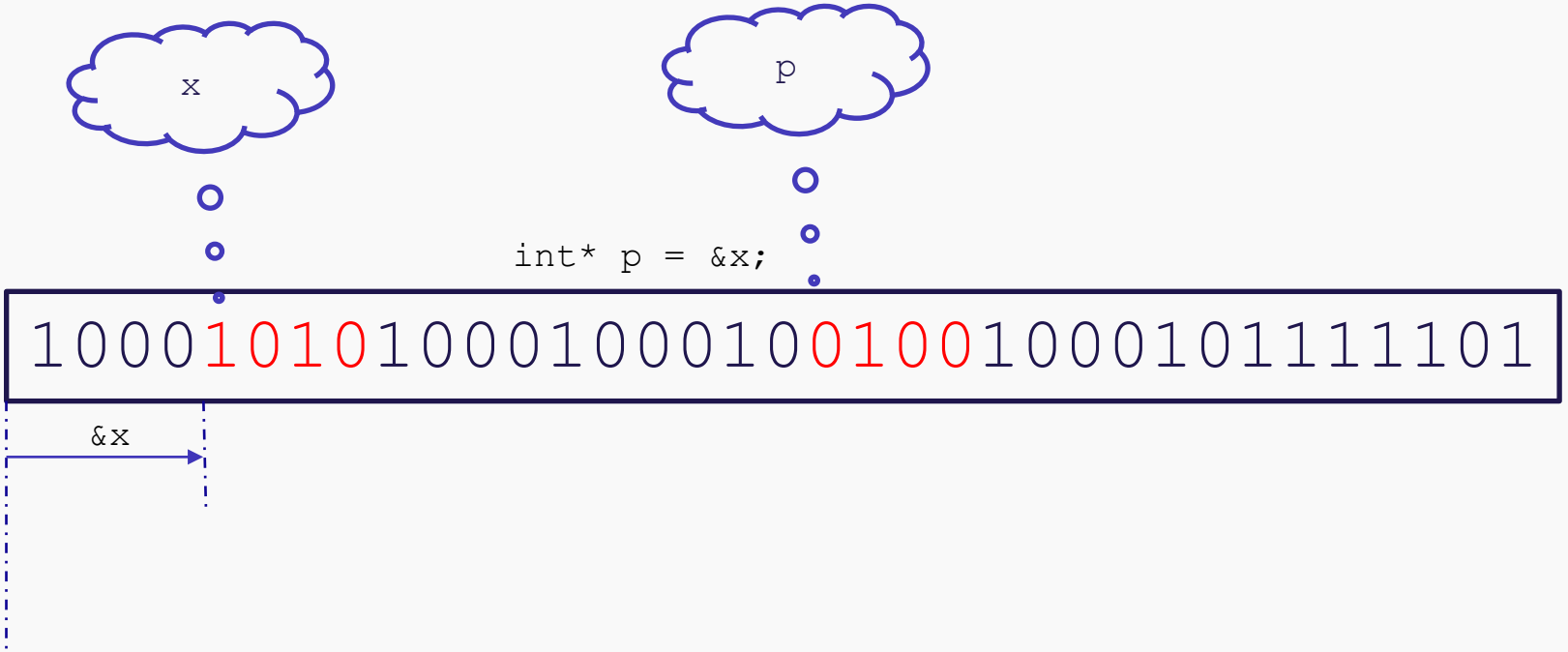
10001010100010001001001000101111101

ПОСМОТРИМ НА ПАМЯТЬ

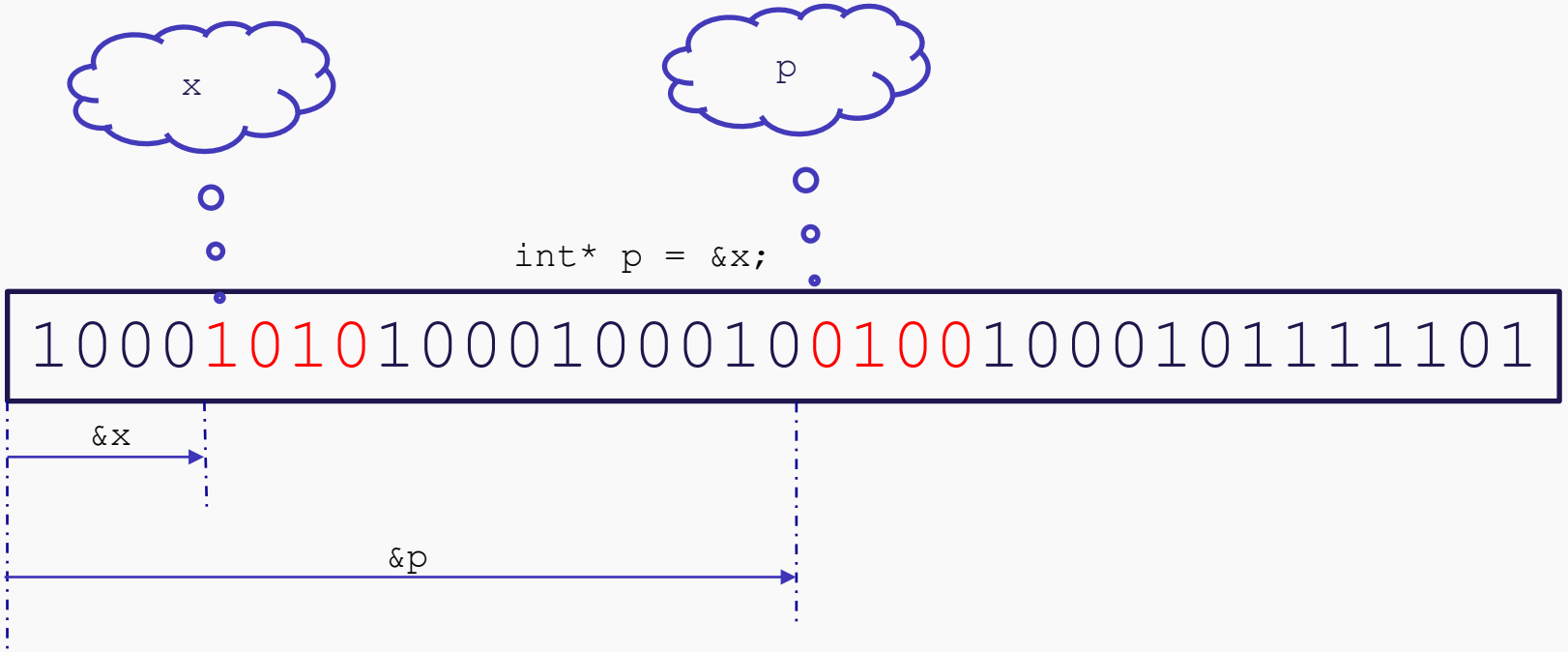
ПОСМОТРИМ НА ПАМЯТЬ



ПОСМОТРИМ НА ПАМЯТЬ



ПОСМОТРИМ НА ПАМЯТЬ



МИНИМУМ ОБ УКАЗАТЕЛЯХ

Чтобы не создавать копию объекта, мы можем хранить его адрес. Переменная, в которой хранится адрес называется указателем.

```
int x = 3;
int* p = &x;
std::cout << p; // ok, увидим адрес x
```

Чтобы обратиться к переменной, на которую указывает p – используем операцию разыменовывания

```
*p = 6;
std::cout << x; // ok, на экране 6
```

ОПАСНО!!!

Опасность подстерегает тех, что использует указатели неосмотрительно. Очень важно понять, что когда вы создаете указатель на C++, то компьютер выделяет память для хранения адреса, но не выделяет памяти для хранения данных, на которые указывает этот адрес. Выделение места для данных выполняется отдельным шагом.

Если пропустить этот шаг, как в следующем фрагменте, то это обеспечит прямой путь к несчастью:

```
long* fellow; // создать указатель на long
*fellow = 223323; // поместить значение в неизвестное место
```

ОПАСНО!!!

Если пропустить этот шаг, как в следующем фрагменте, то это обеспечит прямой путь к несчастью:

```
long* fellow; // создать указатель на long
*fellow = 223323; // поместить значение в неизвестное место
```

Конечно, `fellow` - это указатель. Но на что он указывает? Код не присваивает значения какого-либо адреса переменной `fellow`. Поэтому, куда будет помещено значение 223323? Мы не можем на это ответить. Поскольку переменная `fellow` не была инициализирована, она может иметь какое угодно значение. Что бы в ней ни содержалось, программа будет интерпретировать это как адрес, куда и поместит 223323.

ОПАСНО!!!

Если пропустить этот шаг, как в следующем фрагменте, то это обеспечит прямой путь к несчастью:

```
long* fellow; // создать указатель на long
*fellow = 223323; // поместить значение в неизвестное место
```

Конечно, `fellow` - это указатель. Но на что он указывает? Код не присваивает значения какого-либо адреса переменной `fellow`. Поэтому, куда будет помещено значение 223323? Мы не можем на это ответить. Поскольку переменная `fellow` не была инициализирована, она может иметь какое угодно значение. Что бы в ней ни содержалось, программа будет интерпретировать это как адрес, куда и поместит 223323.

НУЛЕВЫЕ УКАЗАТЕЛИ

Если указатель это просто расстояние, может ли быть нулевое расстояние?

НУЛЕВЫЕ УКАЗАТЕЛИ

Если указатель это просто расстояние, может ли быть нулевое расстояние?
Нулевой указатель – это специальный «маркер ничего». По нему ничего не лежит.
Не надо путать `0`, `NULL` и `nullptr`

```
if (!p) { smth(); } // сработает во всех трёх случаях
```

В языке C++ наш выбор `nullptr` и мы поймем почему это так, когда дойдем до перегрузки функций (которых нет в чистом C и там хватает `NULL`)

ИНДЕКСАЦИЯ УКАЗАТЕЛЕЙ

Изначально указатели всегда были внутрь массива, поэтому поддерживается синтаксис

```
p[2] == *(p + 2);
```

Поскольку сложение коммутативно, `2[p]` тоже сработает

Все ли понимают сколько байт будет добавлено при сложении?

МИНИМУМ О ДИНАМИЧЕСКОЙ ПАМЯТИ

Динамическую память можно выделять и освобождать по необходимости

```
int* p = new int[10]; // не инициализирована
```

Здесь мы выделили память для 10 `int`

```
p[0] = 5; p[9] = 10; // ok, но p[10] будет ошибкой
```

Теперь нам необходимо освободить память

```
delete[] p;
```

В этой точке p никуда не девается, но он указывает в никуда, под ним нет памяти.

ИСПОЛЬЗОВАНИЕ РЕШЕТА

На входе N
На выходе вам предлагается распечатать решето Эратосфена от 2 до N
Печатайте 0, если число простое и 1 – если нет.

Пример: `N = 11`
Выход: `0010101110`

Используйте динамическую память

ДОМАШНЕЕ ЗАДАНИЕ

1. Напишите алгоритм, который использует решето Эратосфена, чтобы найти N-ое простое число
2. Напишите программу, которая раскладывает заданное число N на простые множители и выводит их сумму.
3. Напишите функцию, которая возвращает наибольший общий простой делитель двух чисел.
4. Напишите функцию, которая принимает указатель на массив целых чисел и их размер и возвращает их НОД.
5. Напишите функцию, которая принимает указатель на массив целых чисел и их размер и возвращает их НОК.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Дорохова Т.Ю., Основы алгоритмизации и программирования : учебное пособие для СПО / Т.Ю. Дорохова, И.Е. Ильина. – Саратов, Москва : Профобразование, Ай, Пи Ар Медиа, 2022. – 139 с.
2. Кудинов Ю.И., Основы алгоритмизации и программирования : учебное пособие для СПО / Ю.И. Кудинов, А.Ю. Келина. – 2-е изд. – Липецк, Саратов: Липецкий государственный технический университет, Профообразование, 2020. – 71 с.
3. Дональд Кнут, Искусство программирования. Том 1. Основные алгоритмы / Ю.В. Козаченко. - 3-е изд – Москва, Санкт-Петербург: ВИЛЬЯМС, 2018. – 721 с.