Указатели. Адресная арифметика. Практическое использование указателей. Примеры.

Указатели
Указатель - это переменная, которая хранит адрес памяти другой переменной.
Адрес это участок памяти (ОЗУ), в котором хранится переменная.
Чтобы получить адрес какой либо переменной, нужно приписать к ней слева символ `&`.

- Объявление указателя:

```
int *ptr; // указатель на какую либо переменную int
```

- Присваивание адреса:

```
int a = 10;
int *ptr = &a; // ptr теперь хранит адрес перменной "a".
```

- Разыменовывание (получение значения из адреса):

```
printf("%d\n", *ptr); // Выведет значение переменной a (10)
```

Адресная арифметика
Адресная арифметика позволяет перемещаться по участкам памяти, на которых указывает указатель. ~~(ну да тавтология и чо)~~. Благодаря этому, мы можем взаимодействовать со значениями этих адресов.

- Как пример, операция из предыдущего билета на массивы:

```
int arr[] = {10, 20, 30};
int *ptr = arr; // ptr указывает на первый элемент массива
printf("%d\n", *(ptr + 1)); // Доступ ко второму элементу (20)
```

`ptr + 1` увеличивает адрес на размер одного элемента типа, на который указывает указатель.
Для более глубокого понимания, разберем как это происходит:
Сначала создадим массив и узнаем какие адреса у его элементов:

```
short arr[] = {2,4,8,16,32};
// &arr[0] = 000000557dfffb06
// &arr[1] = 000000557dfffb08
// &arr[2] = 000000557dfffb0a
```

Теперь прибавим к первому адресу 1:
`&arr[0] = 000000557dfffb 06 + 1`
Как и было ранее сказано, Если увеличивать адрес элемента массива на 1, то в памяти этот адрес будет смещаться на размер одного элемента. Получается, если `short` это 2 байта:
`000000557dfffb 06 + 1*(short)`
`000000557dfffb 06 + 2 (byte) = 000000557dfffb 08`
На примере видно как значение адреса увеличилось на два байта.
Надеюсь, стало понятнее.

Практическое использование указателей

1. Изменение значения исходной переменной в функции, потому что указатель в аргументе:

```
 void increment(int *x) {
   (*x)++;
}

int main() {
        int a = 5;          // Было 5
    increment(&a);
    printf("%d\n", a); // Выведет 6
    return 0;
}
```

2. Вывод одномерного массива

```
void printArray(int *arr, int size) {
   for (int i = 0; i < size; i++) {
```

```
            printf("%d ", *(arr + i));
    }
}

int main() {
    int arr[] = {1, 2, 3};
    printArray(arr, 3); // Выведет: 1 2 3
    return 0;
}
```

3. Вывод двумерного массива

```
void printMatrix(int (*matrix)[3], int rows, int cols)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("%d ", *(*(matrix + i) + j));
        }
        printf("\n");
    }
}

int main()
{
    int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
    printMatrix(matrix, 2, 3);   // выведет: 1 2 3
    return 0;                    //          4 5 6
}
```

Лучше в писать в аргументе двумерный массив именно в таком виде `(*matrix)[3]`, это просто лучше)), да и в таком случае, выражения `matrix[i][j]` и `(*(matrix + i) + j)` взаимозаменяемы.