Отличия указателя и массива

Я люблю добрых, поэтому, задавайте вопросы!! Учение - свет, неучение - тьма!

```
int *p;
int q[4];
```

&p = int ** / &q = адрес той же памяти

sizeof(p) = 16 sizeof(q) = зависит от режима компиляции

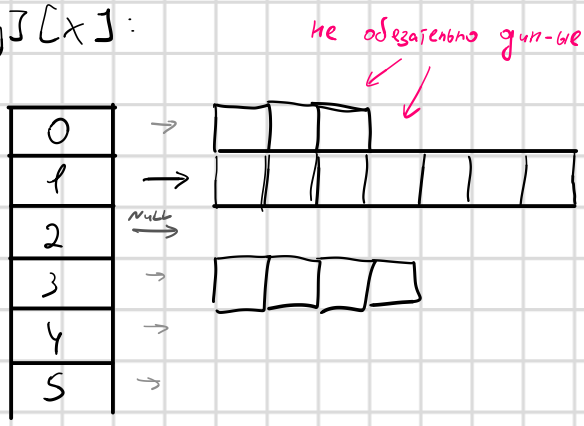
Многомерные массивы и указатели

```
int **p;
```

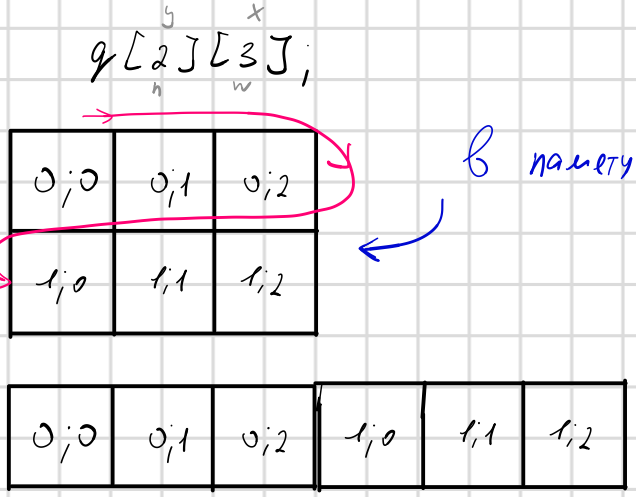
*p - массив

**p - одномерный масив с указателями на int

p[y][x]:




```
int q[2][3];
```



обращение: y*w+x

Пр: **argv

- два обращения к памяти
- перестановка строчек выгодна (переставить указатели)
- одно обращение
- перестановка строчек не выгодна (пробежать по всем)
- кэширование заранее при постоянном шаге

X: p = malloc

for ...

p[y] = malloc

⟶

✓: p = malloc

p[0] = malloc

for ...

p[y] = p[y-1] + ...;

f(int q[2][3]) <=> f(int (*q)[3]) — указатель на массив — либо к.в.к. либо VLA

❗ Не хотим копировать данные при передаче в функцию и просто передаем указатель

f(size_t w, int q[2][w]) <=> f(size_t w, int (*q)[w])

↑ VMT

w, h

int (*q)[w] = malloc(sizeof(int) * w * h); настоящий двумерный массив

q[y][x]

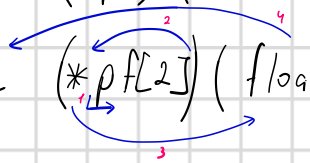
f(size_t w, size_t h, int (*q)[w])

можно int q[][w] координаты нет т.к. превращается в (*q)

int f(float a); функция

int (*pf)(float a); указатель на функцию f

int (*pf[2])(float a); массив из двух указателей на функцию



Пр:

int add(int a, int b);

sub

mul

div

int (* const fop[4])(int, int) = {add, sub, mul, div};

↑ в прототипе имена передаваемых аргументов не важны

```
int compute(uint op, int a, int b)
{
    return fop[op](a, b);
}
```

1) можно применять для передачи расширенной информации (возвращаем успех неудача + что)

2) передавать инф-ию о кол-ве обработанных данных

3) отмена вычислений

Преобразование типов

```
int x = 3.7;
```
double ← неявное приведение

x = 3

```
int x = (int) 3.7;
```
явное (может работать с указателями)

```
float *p;
int x = (int)p
(float *) x;
```
если размерность совпадает или не меньше, то все будет хорошо

будут быть ошибки при любой реализации !!!
(до-до)

```
const float f;
*(int *)&f=3
```
тут норм

будет проблема

Нельзя кастовать структуры и функции (указатели можно)

```
auto x = 3.5f;
```
(не надо)
(в новом стандарте обязаны инициализировать и auto согласует тип со значением)

```
register int x;
```
(нельзя взять адрес &x т.к. в регистрах нет указателей)

можно
typeof(x) y; в C_{23}

Препроцессор

#include ...

не подключение библиотеки (будут быть по голове)

#include <путь> или "путь" ... (содержимое этого файлика вставить сюда)

↑ Тут только пробелы

< > – искать в системе (недрах компилятора)

" " – сначала поищи рядом со мной, потом в системе

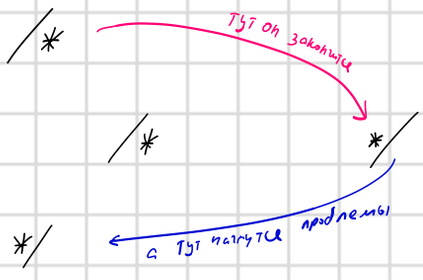
‼ не надо инклудить с файлы (побьют)

#if 0 ← К.В.К.

#if

#else/elif

#endif



#if zzz (неизвестный дефайн) ← автомагом ноль

...

#define ABC int x (заменяй ABC на int x)

\ – эксесткая конкатенация строк

in\
t x;

} int x;

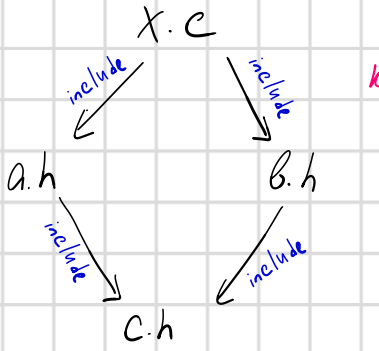
// define ABC(x) int y=x

ABC(3) ⟶ int y=3;

```
#define SUM(x,y) x+y
int x = SUM(2,3)*2; → x = 8
          2+3
```

Как сломать?

```
#define MAX(a,b) ((a)>(b) ?(a):(b))
int x=2, y=3
int z= MAX(x++, y); z=3 x=3 y=3
       x++>y ? x++ : y
#define MAX(a,b) ((a)>(b) ?(a):(b))
int x=2, y=1
int z= MAX(x++, y); z=3 x=4 y=1
       x++>y ? x++ : y
#undef MAX  убираем дефайн
```



Как пофиксить?

```
#ifndef SAMPLE_H (проверяем что есть реф)
#define SAMPLE_H (фиксируем его)
```

Когда во второй раз будем проверять, то скипнем

Или

```
// pragma once (пусть этот файлик включается не более одного раза)
```

Есть еще другие

Слава России!