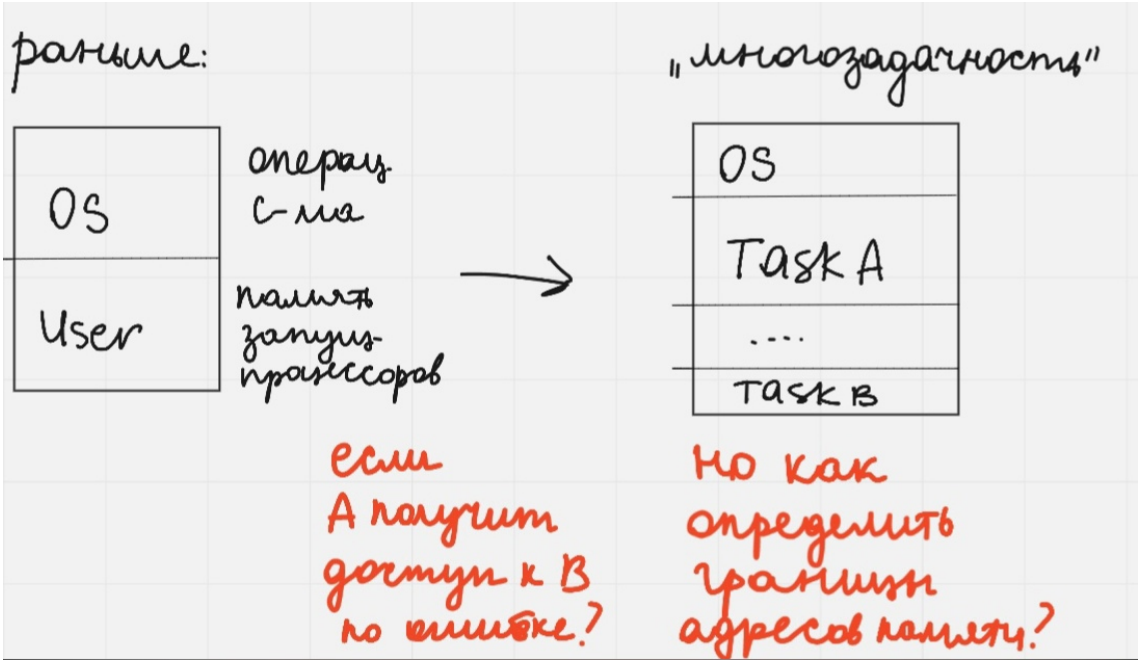
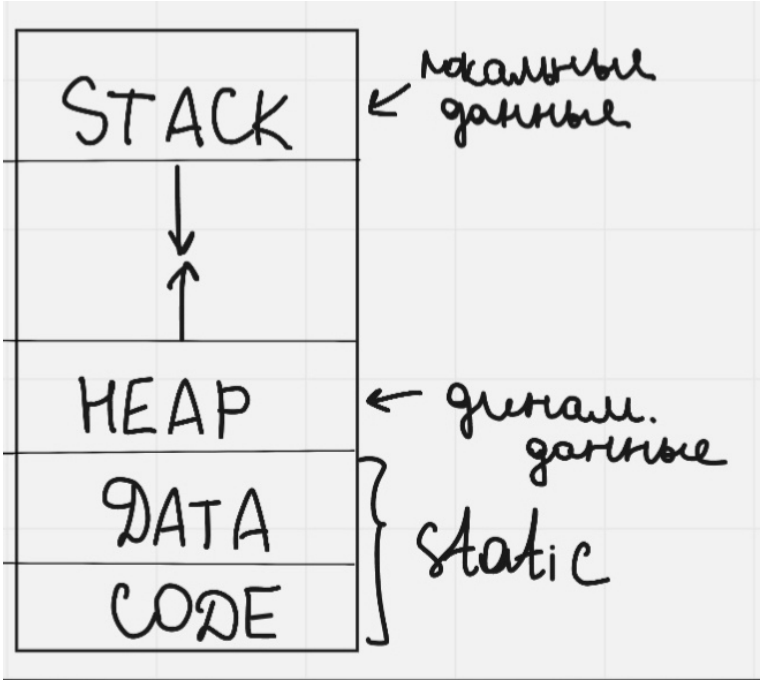
Ответы на контрольные вопросы:

1 вопрос: Из каких сегментов состоит структура памяти процесса?



Каково же решение всех этих красных проблем? Создание адресного пространства.



Code - машинные инструкции, которые процессор должен обработать
Data - данные машинных команд, которе обрабатываются вместе с ними.
Stack - это область памяти, где программа хранит информацию о вызываемых функциях, их аргументах и каждой локальной переменной в функции.
Heap - это область памяти где программист волен делать всё что угодно.
Code - область памяти, где будут храниться инструкции ЦП скомпилированной программы.

2 вопрос: Каким образом связаны встроенные указатели и массивы?

Указатель – переменная, значением которой является адрес ячейки памяти.
Массив - это совокупность определенного количества однотипных переменных, имеющих одно имя (к примеру, int array).
Рассмотрим некоторый статический массив int array[3] = 1,2,3
Для компилятора этот массив является переменной типа int[3]. Мы знаем по отдельности значения каждого из элементов массива, но какое значение у array? Сама переменная array содержит адрес первого элемента массива, как если бы это был указатель.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int array[3] = { 1,2,3 };
    cout << "The array has address " << array << '\n';
    cout << "Element 0 has address: " << &array[0] << '\n';
    return 0;

}
```

```
Консоль отладки Microsoft Visual Studio
The array has address 00EFFDF4
Element 0 has address: 00EFFDF4

C:\Users\shlap\source\repos\First Project\Debug\First Project.exe (процесс 34608) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно…
```

Распространенная ошибка думать, что переменная array и указатель на array являются одним и тем же объектом. Это не так. Хотя оба указывают на первый элемент массива, информация о типе данных у них разная.
Однако есть случаи, когда разница между статическими массивами и указателями имеет значение. Основное различие возникает при использовании оператора sizeof. При использовании в статическом массиве, оператор sizeof возвращает размер всего массива (длина массива умноженная на размер элемента). При использовании с указателем, оператор sizeof возвращает размер адреса памяти (в байтах):

```cpp
#include <iostream>
using namespace std;

int main()
{
    int array[3] = { 1,2,3 };
    cout << sizeof(array) << '\n'; // выведется sizeof(int) * длина array
    int *ptr = array;
    cout << ptr << '\n';
    cout << sizeof(ptr) << '\n'; // выведется размер указателя
    return 0;

}
```

```
Консоль отладки Microsoft Visual Studio
12
00D3FCAC
4

C:\Users\shlap\source\repos\First Project\Debug\First Project.exe (процесс 48756) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно…
```

Размер указателя зависит от разрядности приложения:

на 32-битной версии - 4 байта
на 64-битной версии - 8 байт

4 вопрос: Когда нужно использовать оператор delete [] вместо delete? Какие последствия могут произойти, если забыть написать [] в этом операторе?
(тут я просто поясняю себе за new, потому что потом планирую сохранить эти конспекты)
Динамическое выделение памяти — *это способ запроса памяти из операционной системы запущенными программами по мере необходимости. Эта память не выделяется из ограниченной памяти стека программы, а выделяется из гораздо большего хранилища, управляемого операционной системой — кучи. На современных компьютерах размер кучи может составлять гигабайты памяти.*
Для динамического выделения памяти одной переменной используется оператор new:
Оператор new возвращает указатель, содержащий адрес выделенной памяти.
Для доступа к выделенной памяти создается указатель:
*int *ptr = new int; // динамически выделяем целочисленную переменную и присваиваем её адрес ptr, чтобы затем иметь доступ к ней;*
Затем мы можем разыменовать указатель для получения значения:
ptr = 8; // присваиваем значение 8 только что выделенной памяти
Когда уже всё, что требовалось, выполнено с динамически выделенной переменной — нужно явно указать для C++ освободить эту память. Для переменных это выполняется с помощью оператора delete:
Предположим, что ptr ранее уже был выделен с помощью оператора new:
delete ptr; // возвращаем память, на которую указывал ptr, обратно в операционную систему
ptr = 0; // делаем ptr нулевым указателем (используйте nullptr вместо 0 в C++11)

Оператор delete на самом деле ничего не удаляет. Он просто возвращает память, которая была выделена ранее, обратно в операционную систему. Затем операционная система может переназначить эту память другому приложению (или этому же снова).
Хотя может показаться, что мы удаляем переменную, но это не так! Переменная-указатель по-прежнему имеет ту же область видимости, что и раньше, и ей можно присвоить новое значение, как и любой другой переменной.
Обратите внимание, удаление указателя, не указывающего на динамически выделенную память, может привести к проблемам.
(конец лирического отступления)
В языке программирования C++ оператор delete возвращает память, выделенную оператором new, обратно в кучу. Вызов delete должен происходить для каждого вызова new, чтобы избежать утечки памяти.
Оператор delete [] используется для уничтожения массивов, созданных операцией new []. Использование delete для указателя, возвращаемого new [] или delete [] указателем, возвращаемым new, приводит к поведению undefined. То есть связка new - delete используется для одиночных объектов, а new [] - delete [] - для массивов.