Вопрос 1 Три вида памяти. Работа с кучей на C

- глобальная/статическая память, стек, куча
- `malloc/calloc/realloc/free`
- `void*`

i Глобальная/статическая

Видна везде, определяется вне функций, память выделяется, когда загружается программа.

Listing 1: hello.h

```
#ifndef _hello_H_
#define _hello_H_
extern int a;
#endif _hello_H_
```

Listing 2: main.cpp

```
#include "hello.h"
int a = 0;
int main() {
    return 0;
}
```

Listing 3: helloi.cpp

```
int a=3;
void hello() {
}
```

Static

```
void f() {
static int call_count = 0;
call_count++;
}

int main() {
    f(); f(); f();
}
```

Если объявить *static* вне функции, она не будет видна даже с *extern*.

ii Стек (stack)

Поддерживает операции *push* и *pop*. Когда функция заканчивается, память, выделенная для нее, освобождается. Вычисляется в момент компиляции, как и глобальная.

При запуске рекурсии большой глубины может переполниться *stack* и произойти аварийное завершение.

iii Динамическая (heap)

По запросу программиста во время работы:

```
#include <stdlib.h>
int *p = (int*)malloc(10000*sizeof(int));
p[0] = 1;
free p;
```

typedef позволяет задавать новые типы:

Listing 4: typedef

```
typedef unsigned int size_t
```

Listing 5: malloc

```
void *malloc(size_t size);
```

*void** — указатель на все, универсальный указатель, который можно привести к любому типу, но запрещена адресная арифметика.

```
int *p = (int*)malloc(10000*sizeof(int));
p = (int*)malloc(10*sizeof(int));
```

Особенности динамической памяти

1. Скорость выделения меньше, чем у любой другой
2. Имеет смысл выделять только объекты большого размера. Так как кроме самого объекта нужно создать ссылку на место в памяти.

iv Выделение массива массивов

```
int **m = (int**)malloc(N * sizeof(int*));
for (int i = 0; i < N: ++i) {
    m[i] = (int *)malloc(N * sizeof(int));
}

m[42][42] = 42;

for (int i = 0; i < N; ++i) {
    free(m[i]);
}
free(m);
```

v Еще

- `calloc`

  — выделяет память и инициализирует нулями

- `realloc`

  — изменяет размер существующего массива:

  1. если нужное число байт не занято рядом, просто увеличиваем
  2. иначе находим другое место и переносим туда весь массив
  3. если нет вообще памяти, возвращаем 0.