-3- Три вида памяти. Работа с кучей

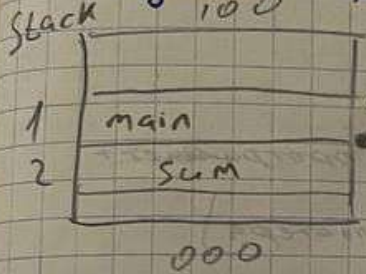
Память:

1гб	OS	- операц. система
10мб	stack, растёт ↓	- локал. переменные, параметры функций, возвращаемое значение
~2,9гб	Heap, растёт ↑	- Динамическая память
10мб	static variables	- глобальные переменные, существуют всё время программы
10мб	Двоич. код	

0 - адрес ↑

Stack

- Локал переменные функции, параметры
- Код выделение и освобождение генерирует компилятор (выделили память / 3 - удалили память)
- Выделяется при входе в память, удаляется при выходе



SP - хранит адрес в стеке, где начинается свободное

SP = -24

⇓

выделили память; освобождение - сложение

Кадр стека:

кадр main	c, b, e, RV, RA
кадр sum	a, b, c, RV, RA

RV - возвращаемое значение

RA - на какой адрес вернуться

Рекурсия плохо потому что стек может закончиться

Глобальная память

- глобальные переменные (вне функции), статические (static)
- код для выделение и освобождение генерирует компилятор
- выделяется при загрузке в память, освобождается при завершении программы
- глобальные перемен. непонятно в каком порядке инициализируются, статические — при входе в функцию

☹ Глобал. переменных

*1 Потенциальный конфликт имён при большом коде

*2 Трудно анализировать

*3 Неудобен порядок инициализации -> нарушение работы кода →

! Глобальные перемен = 0 автоматически !

// a.cpp : int ten = 10; - первый инициализ., т.к. константа
int x = ten; - второе / третье
① ②

int return_num() { return x; }

// b.cpp : int y = return_num(); - третье / второе

① x = 10 → y = 10 (!?)

② y = 0 → x = 10

Объявление переменных:

Глобал

int a = 0; - будет во всех файлах
void f() { } при помощи extern

1.h: extern int last_rand; - объявим
1.cpp: int last_rand = 0; - выделили память
void rand() { }
2.cpp: #include "1.h"
int rand() { last_rand... }

Static глобал, но видна только в этом файле

static int b = 0;
void f() {
static int c = 0; - статическая
видна только в функции }

Куча

- код для выделения и освобождения пишет программист
- нет ограничений по размеру (только память компьютера)

malloc / calloc / realloc / free — stdlib.h

malloc - функция, обращающаяся к операционной системе с просьбой выделить место (непрерывный кусок!) в куче;
если выделение прошло - возвращает указатель на память
иначе - NULL
» долго, пробегается по всем свободным кусочкам

int* p = (int*) malloc (100 * sizeof(int)) (изначально malloc возвращает void*)

free - освобождает память

free(p); (обязательно отчищать память после выделения самостоятельно!)

void* calloc (size_t num, size_t size) - выделяет num эл. по size и заполняет „0"

void* realloc (void *ptr, size_t new_size) - выделяет новую память или дополняет к ptr : new_size - size;
p = realloc (p, c * sizeof(int)); free(p);

void *
Нельзя
-1
использ

void * - указатель на любой тип данных

Нельзя использовать адресную арифметику, т.к. компилятор не знает тип.

-1
используют, например, в универсальных шаблонах