Семинар 6. Леденцов P32081

TASK 1

/\* heap-0.c \*/

#include <stdbool.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#define heap\_blocks 16

#define block\_capacity 1024

/\*структура кучи

  содержит внутри себя массив блоков символов

  и bool массив, показывающий какие блоки заняты.

\*/

struct heap {

    struct block {

        char contents[block\_capacity];

    } blocks[heap\_blocks];

    bool is\_occupied[heap\_blocks];

} global\_heap = {0}; // struct heap global\_heap = {0}; просто инициализация кучи

/\*структура представляющая собой информацию о блоке

 \* содержит данные о его номере, валиден ли и ссылку

 \* на саму структуру, где он лежит

\*/

struct block\_id {

    size\_t value;

    bool   valid;

    struct heap\* heap;

};

/\*

 \* инициализирует блок в куче

\*/

struct block\_id block\_id\_new(size\_t value, struct heap\* from) {

    return (struct block\_id){.valid = true, .value = value, .heap = from};

}

/\*

 \* возвращает невалидный блок

\*/

struct block\_id block\_id\_invalid() { return (struct block\_id){.valid = false}; }

/\*

 \* проверка блока на валидность(проверка размера)

\*/

bool block\_id\_is\_valid(struct block\_id bid) {

    return bid.valid && bid.value < heap\_blocks;

}

/\*

 \* проверка блока в куче на то, что он свободен

\*/

bool block\_is\_free(struct block\_id bid) {

    // пройтись по массиву is\_occupied и проверить, что блок по заданному id свободен

    return bid.heap->is\_occupied[bid.value];

}

/\* allocate \*/

/\* find a free block, reserve it and return its id \*/

/\*

 \* пробегается по куче(is\_occupied) в поисках свободного блока, и если

 \* находит таковой - инициализирует его

 \* иначе, возвращает невалидный блок(block\_id)

\*/

struct block\_id block\_allocate(struct heap\* heap) {

    for(size\_t i = 0; i<heap\_blocks; i++){

        if(heap->is\_occupied[i]==false){

            heap->is\_occupied[i]=true;

            return block\_id\_new(i, heap);

        }

    }

    return block\_id\_invalid();

}

/\* mark block as 'free' \*/

/\*освобождает блок в куче

 \* просто в is\_occupied пометить соответствующий элемент

 \* массива как false и поместить нули в соответствующий блок кучи

\*/

void block\_free(struct block\_id b) {

    b.heap->is\_occupied[b.value] = false;

}

/\* printer \*/

const char\* block\_repr(struct block\_id b) {

    static const char\* const repr[] = {[false] = " .", [true] = " ="};

    if (b.valid)

        return repr[b.heap->is\_occupied[b.value]];

    else

        return "x";

}

void block\_debug\_info(struct block\_id b, FILE\* f) {

    fprintf(f, "%s", block\_repr( b));

}

void block\_foreach\_printer(struct heap\* h, size\_t count,

                           void  printer(struct block\_id, FILE\* f),

                           FILE\* f) {

    for (size\_t c = 0; c < count; c++)

        printer(block\_id\_new(c,h), f);

}

void heap\_debug\_info(struct heap\* h, FILE\* f) {

    block\_foreach\_printer(h, heap\_blocks, block\_debug\_info, f);

    fprintf(f, "\n");

}

/\*  -------- \*/

int main() {

    heap\_debug\_info(&global\_heap, stdout);//печатаем содержимое кучи в поток вывода

    block\_allocate(&global\_heap);//зарезервировали место в куче(блок)

    struct block\_id bid = block\_allocate(&global\_heap);//ещё раз сделали то же самое, но в переменную

    block\_allocate(&global\_heap);//аналогично

    block\_free(bid);//освобождаем блок

    heap\_debug\_info(&global\_heap, stdout);//печатаем содержимое кучи в stdout 1 и 3 блоки должны быть заняты

    return 0;

}

TASK 2

/\* heap-1.c \*/

#include <stdbool.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#define HEAP\_BLOCKS 16

#define BLOCK\_CAPACITY 1024

/\*

 \* статус блока: свободен, единичный блок, затрагивает первый блок в куче, блок не затрагивающий первый и последний блок, затрагивает последний блок в куче

 \* \*/

enum block\_status {

    BLK\_FREE = 0, BLK\_ONE, BLK\_FIRST = 1, BLK\_CONT, BLK\_LAST

};

/\*наша куча теперь имеет большее количество статусов вместо true или false

 \* \*/

struct heap {

    struct block {

        char contents[BLOCK\_CAPACITY];

    } blocks[HEAP\_BLOCKS];

    enum block\_status status[HEAP\_BLOCKS];

} global\_heap = {0};

struct block\_id {

    size\_t value;

    bool valid;

    struct heap \*heap;

};

struct block\_id block\_id\_new(size\_t value, struct heap \*from) {

    return (struct block\_id) {.valid = true, .value = value, .heap = from};

}

struct block\_id block\_id\_invalid() {

    return (struct block\_id) {.valid = false};

}

bool block\_id\_is\_valid(struct block\_id bid) {

    return bid.valid && bid.value < HEAP\_BLOCKS;

}

/\* Find block \*/

bool block\_is\_free(struct block\_id bid) {

    if (!block\_id\_is\_valid(bid))

        return false;

    return bid.heap->status[bid.value] == BLK\_FREE;

}

/\* Allocate \*/

/\*Выделить память в куче под определённое количество блоков

 \* посчитать ближайшие блок в куче, начиная с которого будет свободно size блоков

 \* заполнить статус для всех блоков, которые мы резервируем - первый по порядку FIRST и последний LAST, остальные CONT

 \*

 \* \*/

struct block\_id block\_allocate(struct heap \*heap, size\_t size) {

    size\_t free\_blocks\_counter = 0;

    size\_t first\_block = -1;

    if (size < HEAP\_BLOCKS) {

        for (size\_t i = 0; i < HEAP\_BLOCKS; i++) {

            if (heap->status[i] == BLK\_FREE) {

                free\_blocks\_counter += 1;

                if (free\_blocks\_counter == size) {

                    first\_block = i - size + 1;//

                    break;

                }

            } else {

                free\_blocks\_counter = 0;

            }

        }

        if (first\_block != -1) {

            for (size\_t i = first\_block; i < first\_block + size; i++) {

                if (i == first\_block) {

                    heap->status[i] = BLK\_FIRST;

                } else if (i == first\_block + size - 1) {

                    heap->status[i] = BLK\_LAST;

                } else {

                    heap->status[i] = BLK\_CONT;

                }

            }

            return block\_id\_new(first\_block, heap);

        } else {

            return block\_id\_invalid();//недостаточно памяти в куче для выделения блока

        }

    }

    return block\_id\_invalid();//превышает размер кучи

}

/\* Free \*/

// ? ? ?

void block\_free(struct block\_id b) {

    if (b.heap->status[b.value] == BLK\_FIRST) {

        size\_t index = b.value; //адрес первого элемента в массиве

        while (b.heap->status[index] != BLK\_LAST) {

            b.heap->status[index] = BLK\_FREE;

            index += 1;

        }

        b.heap->status[index] = BLK\_FREE;

    } else {

        printf("некорректное освобождение памяти в куче");

    }

}

/\* Printer \*/

const char \*block\_repr(struct block\_id b) {

    static const char \*const repr[] = {[BLK\_FREE] = " .",

            [BLK\_ONE] = " \*",

            [BLK\_FIRST] = "[=",

            [BLK\_LAST] = "=]",

            [BLK\_CONT] = " ="};

    if (b.valid)

        return repr[b.heap->status[b.value]];

    else

        return "INVALID";

}

void block\_debug\_info(struct block\_id b, FILE \*f) {

    fprintf(f, "%s", block\_repr(b));

}

void block\_foreach\_printer(struct heap \*h, size\_t count,

                           void printer(struct block\_id, FILE \*f), FILE \*f) {

    for (size\_t c = 0; c < count; c++)

        printer(block\_id\_new(c, h), f);

}

void heap\_debug\_info(struct heap \*h, FILE \*f) {

    block\_foreach\_printer(h, HEAP\_BLOCKS, block\_debug\_info, f);

    fprintf(f, "\n");

}

int main() {

    heap\_debug\_info(&global\_heap, stdout);//печатаем содержимое кучи в поток вывода

    block\_allocate(&global\_heap, 3);//зарезервировали место в куче(блок)

    struct block\_id bid = block\_allocate(&global\_heap, 10);//ещё раз сделали тоже самое, но в переменную

    //block\_allocate(&global\_heap);//аналогично

    heap\_debug\_info(&global\_heap, stdout);

    block\_free(bid);//освобождаем блок

    heap\_debug\_info(&global\_heap, stdout);//печатаем содержимое кучи в stdout 1 и 3 блоки должны быть заняты

}

TASK 3

VECTOR.h:

#include <stdio.h>

#ifndef VECTOR\_H

#define VECTOR\_H

#define VECTOR\_INIT\_CAPACITY 4

typedef struct vector {

    void \*\*items;

    int capacity;

    int total;

} vector;

typedef void\* (\*callback)(void\* data);

void vector\_init(vector \*);

int vector\_total(vector \*);

void vector\_resize(vector \*, int);

void vector\_add(vector \*, void \*);

void vector\_set(vector \*, int, void \*);

void \*vector\_get(vector \*, int);

void vector\_delete(vector \*, int);

void vector\_free(vector \*);

void vector\_foreach(vector\* v, callback f);

void print\_vector(vector\* v, FILE\* f, callback printer);

#endif

VECTOR.C

#include <stdio.h>

#include <stdlib.h>

#include "vector.h"

void vector\_init(vector \*v)

{

    v->capacity = VECTOR\_INIT\_CAPACITY;

    v->total = 0;

    v->items = malloc(sizeof(void \*) \* v->capacity);

}

int vector\_total(vector \*v)

{

    return v->total;

}

void vector\_resize(vector \*v, int capacity)

{

    void \*\*items = realloc(v->items, sizeof(void \*) \* capacity);

    if (items) {

        v->items = items;

        v->capacity = capacity;

    }

}

void vector\_add(vector \*v, void \*item)

{

    if (v->capacity == v->total)

        vector\_resize(v, v->capacity \* 2);

    v->items[v->total++] = item;

}

void vector\_set(vector \*v, int index, void \*item)

{

    if (index >= 0 && index < v->total)

        v->items[index] = item;

}

void \*vector\_get(vector \*v, int index)

{

    if (index >= 0 && index < v->total)

        return v->items[index];

    return NULL;

}

void vector\_delete(vector \*v, int index)

{

    if (index < 0 || index >= v->total)

        return;

    v->items[index] = NULL;

    for (int i = index; i < v->total - 1; i++) {

        v->items[i] = v->items[i + 1];

        v->items[i + 1] = NULL;

    }

    v->total--;

    if (v->total > 0 && v->total == v->capacity / 4)

        vector\_resize(v, v->capacity / 2);

}

void vector\_free(vector \*v)

{

    free(v->items);

}

typedef void\* (\*callback)(void\* data);

void vector\_foreach(vector\* v, callback f){

     for (int i = 0; i < vector\_total(v); i++) f(vector\_get(v,i));

}

void print\_vector(vector\* v, FILE\* f, callback to\_string){

    for (int i = 0; i < vector\_total(v); i++){

        char\* str = to\_string(vector\_get(v,i));

        fprintf(f, "%s  ", str);

    }

}

MAIN.C

#include <stdio.h>

#include <stdlib.h>

#include "vector.h"

void\* to\_str(void\* data){

    return data;

}

int main(void)

{

    vector v;

    vector\_init(&v);

    vector\_add(&v, "костя");

    vector\_add(&v, "лох");

    vector\_add(&v, "объелся");

    vector\_add(&v, "блох");

    FILE \*file;

    file = fopen("f.txt", "w");

    if (file == NULL){

        printf("Error opening file!\n");

        exit(1);

    }

    print\_vector(&v,file, to\_str);

    fclose(file);

    print\_vector(&v,stdout, to\_str);

    printf("\n");

    vector\_free(&v);

}

Изображение выглядит как текст, стол

Автоматически созданное описание