Отчет по лабораторной работе № 25+26 по курсу
Алгоритмы и структуры данных

Студент группы М8О-103Б-22 Ахметшин Булат Рамилевич, № по списку 2

Контакты www, e-mail, icq, skype ahmbulat04@yandex.ru

Работа выполнена: 31.05.2023 г.

Преподаватель: доцент каф. 806 Никулин С.П.

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 202 __ г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Автоматизация сборки программ модульной структуры на языке Си с использованием утилиты make. Абстрактные типы данных. Рекурсия. Модульное программирование на языке Си. _____

2. **Цель работы:** Научиться пользоваться утилитой make. Научиться составлять линейные структуры данных и реализовывать на них алгоритмы. _____

3. **Задание** (*вариант № 3, 3*): Составить makefile для компиляции программы с реализацией дека и функций для него. ____

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____ с ОП _____ Мб,
НМД _____ Мб. Терминал _____ адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel(R) Core(TM) i7-10510U____ с ОП 8 ГБ____ НМД SSD 512 ГБ____ . Монитор Встроенный 1920x1080
Другие устройства _____

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX_____, наименование Ubuntu_____ версия 22.04_____
интерпретатор команд GNU bash_____ версия 5.1.16_____
Система программирования Visual Studio Code_____ версия 1.77.3_____
Редактор текстов Sublime Text 3_____ версия 3211_____
Утилиты операционной системы Стандартные утилиты OS Linux_____
Прикладные системы и программы Редактор текста nano._____

Местонахождение и имена файлов программ и данных на домашнем компьютере
/home/bulat/Studying/prprm/l/l24/ty2

6. **Идея, метод, алгоритм** решение задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

Создать makefile, который отдельно компилирует deque.c и main.c в объектные файлы, также отдельную команду для очистки от объектных файлов и исполняемого файла.

7. **Сценарий выполнения работы** (план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты либо соображения по тестированию)

Составить makefile, написать реализацию дека и метода и процедуры на основе этого дека, соответствующих моему варианту.

Пункты 1-7 отчета составляются строго до начала лабораторной работы.
Допущен к выполнению работы. **Подпись преподавателя** _____

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)

```
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ ls
deque.c  deque.h  l25-2012.djvu  l26-2012.djvu  logs  main.c  makefile
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ cat makefile
CC = gcc
CFLAGS = -std=c99 -Wall -Wextra

main: main.o deque.o
    $(CC) $(CFLAGS) -o main main.o deque.o
main_o:
    $(CC) $(CFLAGS) -c main.c
deque_o:
    $(CC) $(CFLAGS) -c deque.c
clean:
    rm -f *.o main
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ cat deque.h
#ifndef _UDT_H
#define _UDT_H

#include <inttypes.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

typedef int64_t type;
typedef uint64_t size_t;

typedef struct {
    // Pointer to allocated memory
    type* p;
    // Pointer to the firest element of the deque
    type* begin;
    // Pointer to the last element of the deque
    type* back;
    // Pointer to place right after back
    type* end;
    // Count of elements in deque
    size_t size;
    // Count of elements allocated in memory
    size_t memo;
} deque;

// Constructs deque with diven length
void deque_create(deque*, size_t);

// Constructs deque with diven length
void deque_create_fill(deque*, size_t, type);

// Constructs empty deque
void deque_create_empy(deque*);

// Construct deque from given array
void deque_construct(deque*, type*, size_t);

// Copy deque from other given deque
void deque_copy(deque*, deque* const);

// Checks if deque is empty
bool deque_is_empty(deque* const);
// Checks if deque is allocated
bool deque_is_allocated(deque* const);

void deque_clear(deque*);

void deque_free(deque*);

// Push and pop from each side
void deque_push_front(deque*, type);
void deque_push_back(deque*, type);
type deque_pop_front(deque*);
type deque_pop_back(deque*);

void deque_fill(deque*, type);

// Print deque in console
void deque_print(deque*, char* (*)(type));

// Size of deque
size_t deque_size(deque* const);
```

```
type* deque_begin(deque* const);
type* deque_back(deque* const);
type* deque_end(deque* const);

// Insert
bool deque_insert(deque*, size_t, type);
bool deque_erase(deque*, size_t);

// Filters
deque deque_filter(deque*, type, bool (*)(type, type));

// Concatenate deques

deque deque_concatenate(deque* const, deque* const);

// With iterating
void hoars_qsort_iter(type*, type*, bool (*)(type, type));

// With concatenating left and right sorted deqs
deque hoars_qsort_conc(deque, bool (*)(type, type), bool (*)(type, type));

#define swap(a, b, _type) \
{                         \
    _type c = a;          \
    a = b;                \
    b = c;                \
}

#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

type average(type, type, type, bool (*)(type, type));

char* int64_t_to_str(int64_t);

bool int64_t_less(int64_t, int64_t);
bool int64_t_nmore(int64_t, int64_t);
bool int64_t_more(int64_t, int64_t);
bool int64_t_nless(int64_t, int64_t);
bool int64_t_eq(int64_t, int64_t);
bool int64_t_neq(int64_t, int64_t);

#endif
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ cat deque.c
#include "deque.h"

void deque_create_empy(deque* d) {
    d->p = (type*)calloc(1, sizeof(type));
    d->size = 0;
    d->memo = 1;
    d->begin = d->back = d->end = NULL;
}

void deque_create(deque* d, size_t l) {
    if (l == 0) {
        deque_create_empy(d);
    }
    d->p = (type*)calloc(l, sizeof(type));
    d->size = 0;
    d->memo = l;
    d->begin = d->back = d->end = NULL;
}

void deque_create_fill(deque* d, size_t l, type f) {
    if (l == 0) {
        deque_create_empy(d);
    }
    d->p = (type*)malloc(sizeof(type) * l);
    for (size_t i = 0; i < l; ++i) {
        d->p[i] = f;
    }
    d->size = l;
    d->memo = l;
    d->begin = d->p;
    d->back = d->begin + d->size - 1;
    d->end = d->back + 1;
}

void deque_construct(deque* d, type* a, size_t l) {
    d->p = (type*)calloc(l, sizeof(type));
    d->size = l;
    d->memo = d->size;
    for (size_t i = 0; i < l; ++i) {
        d->p[i] = a[i];
    }
```

```
    d->begin = d->p;
    d->back = d->begin + d->size - 1;
    d->end = d->back + 1;
}

void deque_copy(deque* d, deque* const o) {
    d->p = (type*)calloc(o->memo, sizeof(type));
    d->size = o->size;
    d->memo = o->memo;
    for (size_t i = 0; i < o->size; ++i) {
        d->p[i] = o->p[i];
    }
    d->begin = d->p;
    d->back = d->begin + d->size - 1;
    d->end = d->back + 1;
}

bool deque_is_empty(deque* const d) {
    return d->begin == NULL && d->back == NULL;
}

bool deque_is_allocated(deque* const d) {
    return d->p == NULL;
}

void deque_clear(deque* d) {
    deque_fill(d, (type)(0));
    d->begin = d->back = d->end = NULL;
    d->size = 0;
}

void deque_free(deque* d) {
    if (!deque_is_empty(d)) {
        free(d->p);
    }
    d->p = d->begin = d->back = d->end = NULL;
    d->size = d->memo = 0;
}

void deque_push_front(deque* d, type x) {
    type* _p = d->p;
    bool allocated = false;
    if (d->size >= d->memo) {
        d->memo *= 2;
        _p = (type*)calloc(d->memo, sizeof(type));
        allocated = true;
    }
    for (size_t i = 0; i < d->size; ++i) {
        _p[i + 1] = d->p[i];
    }
    _p[0] = x;
    if (allocated) {
        free(d->p);
    }
    d->p = _p;
    d->size++;
    d->begin = d->p;
    d->back = d->p + d->size - 1;
    d->end = d->back + 1;
}

void deque_push_back(deque* d, type x) {
    type* _p = d->p;
    bool allocated = false;
    if (d->size >= d->memo) {
        d->memo *= 2;
        _p = (type*)calloc(d->memo, sizeof(type));
        allocated = true;
    }
    for (size_t i = 0; i < d->size; ++i) {
        _p[i] = d->p[i];
    }
    _p[d->size] = x;
    if (allocated) {
        free(d->p);
    }
    d->p = _p;
    d->size++;
    d->begin = d->p;
    d->back = d->p + d->size - 1;
    d->end = d->back + 1;
}

type deque_pop_front(deque* d) {
    type x = (type)(0);
```

```
    if (d->size > 0) {
        x = d->p[0];
        for (size_t i = 0; i < d->size - 1; ++i) {
            d->p[i] = d->p[i + 1];
        }
        *(d->back) = (type)(0);
    }
    if (d->size > 1) {
        d->back--;
        d->end--;
        d->size--;
    } else {
        deque_clear(d);
    }
    return x;
}

type deque_pop_back(deque* d) {
    type x = (type)(0);
    if (d->size > 0) {
        x = *(d->back);
        *(d->back) = (type)(0);
    }
    if (d->size > 1) {
        d->back--;
        d->end--;
        d->size--;
    } else {
        deque_clear(d);
    }
    return x;
}

void deque_fill(deque* d, type x) {
    for (size_t i = 0; i < d->size; ++i) {
        d->p[i] = x;
    }
}

void deque_print(deque* d, char* (*to_str)(type)) {
    printf("{");
    if (d->size > 0) {
        char* p;
        for (size_t i = 0; i < d->size - 1; ++i) {
            p = to_str(d->p[i]);
            printf(" %s,", p);
            free(p);
        }
        p = to_str(*(d->back));
        printf(" %s ", p);
        free(p);
    }
    printf("}");
}

size_t deque_size(deque* const d) {
    return d->size;
}

type* deque_begin(deque* const d) {
    return d->begin;
}

type* deque_back(deque* const d) {
    return d->back;
}

type* deque_end(deque* const d) {
    return d->end;
}

bool deque_insert(deque* d, size_t index, type x) {
    if (d->size < index + 1) {
        fprintf(stderr, "%s", "Error: trying to insert element out of deque bounds!\n");
        return false;
    }
    if (d->size < d->memo) {
        for (size_t i = d->size; i > index; --i) {
            d->p[i] = d->p[i - 1];
        }
        d->p[index] = x;
    } else {
        d->memo *= 2;
        type* _p = (type*)calloc(d->memo, sizeof(type));
        for (size_t i = 0; i < d->size; ++i) {
```

```
            if (i < index) {
                _p[i] = d->p[i];
            } else {
                _p[i + 1] = d->p[i];
            }
        }
        _p[index] = x;
        free(d->p);
        d->p = _p;
    }
        d->size++;
    d->begin = d->p;
    d->back = d->begin + d->size - 1;
    d->end = d->back + 1;
    return true;
}

bool deque_erase(deque* d, size_t index) {
    if (d->size < index + 1) {
        fprintf(stderr, "%s", "Error: trying to erase element out of deque bounds!\n");
        return false;
    } else {
        if (d->size > 0) {
            for (size_t i = index; i < d->size - 1; ++i) {
                d->p[i] = d->p[i + 1];
            }
            *(d->back) = (type)(0);
        }
        if (d->size > 1) {
            d->back--;
            d->end--;
            d->size--;
        } else {
            deque_clear(d);
        }
    }
    return true;
}

deque deque_filter(deque* d, type x, bool (*cmp)(type, type)) {
    deque r;
    deque_create_empy(&r);
    for (size_t i = 0; i < d->size; ++i) {
        if (cmp(d->p[i], x)) {
            deque_push_back(&r, d->p[i]);
        }
    }
    return r;
}

deque deque_concatenate(deque* const a, deque* const b) {
    deque d;
    deque_create_empy(&d);
    d.p = (type*)calloc(a->size + b->size, sizeof(type));
    d.size = a->size + b->size;
    d.memo = d.size;

    for (size_t i = 0; i < a->size; ++i) {
        d.p[i] = a->p[i];
    }
    for (size_t i = a->size; i < d.size; ++i) {
        d.p[i] = b->p[i - a->size];
    }
    d.begin = d.p;
    d.back = d.begin + d.size - 1;
    d.end = d.back + 1;
    return d;
}

void hoars_qsort_iter(type* begin, type* end, bool (*cmp)(type, type)) {
    if (end - begin > 1) {
        // Pick pivot
        type* back = end - 1;
        type* mid = begin + (back - begin) / 2;
        if (cmp(*mid, *begin)) {
            swap(*mid, *begin, type);
        }
        if (cmp(*back, *begin)) {
            swap(*back, *begin, type);
        }
        if (cmp(*back, *mid)) {
            swap(*back, *mid, type);
        }
        type pivot = *mid;
        // Hoar's partition scheme
```

```
        type* i = begin,* j = back;
        while (true) {
            while (j - i > 0 && cmp(*i, pivot)) {
                ++i;
            }
            while (j - i > 0 && cmp(pivot, *j)) {
                --j;
            }
            if (i - j >= 0) {
                break;
            }
            swap(*i, *j, type);
            ++i; --j;
        }
        hoars_qsort_iter(begin, j, cmp);
        hoars_qsort_iter(j + 1, end, cmp);
    }
}

deque hoars_qsort_conc(deque d, bool (*less)(type, type), bool(*nless)(type, type)) {
    if (d.size <= 1) {
        return d;
    }
    type pivot = average(*d.begin, *(d.begin + (d.back - d.begin) / 2), *d.back, less);
    deque l = hoars_qsort_conc(deque_filter(&d, pivot, less), less, nless);
    deque r = hoars_qsort_conc(deque_filter(&d, pivot, nless), less, nless);
    deque res = deque_concatenate(&l, &r);
    deque_free(&l); deque_free(&r);
    return res;
}

type average(type a, type b, type c, bool (*less)(type, type)) {
    if (less(a, b)) {
        if (less(b, c)){
            return b;
        } else {
            return c;
        }
    } else if (less(a, c)) {
        return a;
    } else if (less(b, c)) {
        return c;
    } else {
        return b;
    }
}

char* int64_t_to_str(int64_t x) {
    char* p;
    if (x == 0) {
        p = (char*)malloc(sizeof(char) * 2);
        p[0] = '0';
        p[1] = '\0';
    }
    else {
        int64_t _x = x;
        size_t l = 0;
        while (_x != 0) {
            _x /= 10;
            ++l;
        }
        if (x < 0) {
            p = (char*)malloc(sizeof(char) * (l + 2));
            p[0] = '-';
            ++l;
            x = -x;
        } else {
            p = (char*)malloc(sizeof(char) * (l + 1));
        }
        p[l] = '\0';
        while (x != 0) {
            p[--l] = (char)((x % 10) + 48);
            x /= 10;
        }
    }

    return p;
}

bool int64_t_less(int64_t a, int64_t b) {
    return a < b;
}

bool int64_t_nmore(int64_t a, int64_t b) {
    return a <= b;
```

```
}

bool int64_t_more(int64_t a, int64_t b) {
    return a > b;
}

bool int64_t_nless(int64_t a, int64_t b) {
    return a >= b;
}

bool int64_t_eq(int64_t a, int64_t b) {
    return a == b;
}

bool int64_t_neq(int64_t a, int64_t b) {
    return a != b;
}

bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ cat main.c
#include <stdio.h>
#include <inttypes.h>

#include "deque.h"


int main() {


    deque d;

    deque_create_empy(&d);

    for (size_t i = 0; i < 10; ++i) {
        deque_push_back(&d, i * i);
    }

    printf("Push back illustration:\n");
    deque_print(&d, int64_t_to_str);
    printf("\n");


    deque_insert(&d, 5, -10);

    printf("Insert illustration:\n");
    deque_print(&d, int64_t_to_str);
    printf("\n");


    printf("Pop back illustration:\n");
    for (size_t i = d.size; i > 0; --i) {
        deque_pop_back(&d);
        deque_print(&d, int64_t_to_str);
        printf("\n");
    }

    const size_t sz = 5;

    type a[sz];

    for (size_t i = 0; i < sz; ++i) {
        a[i] = i + 1;
    }

    deque_free(&d);

    deque_construct(&d, a, sz);

    printf("Construct from array illustration:\n");
    deque_print(&d, int64_t_to_str);
    printf("\n");

    deque_fill(&d, 9);

    printf("Fill deque illustration:\n");
    deque_print(&d, int64_t_to_str);
    printf("\n");

    d.p[3] = 8;

    printf("Erase element from deque illustration:\n");
    deque_print(&d, int64_t_to_str);
    printf("\n");

    deque_erase(&d, 3);
```

```
    deque_print(&d, int64_t_to_str);
    printf("\n");

    deque r;

    deque_create(&r, sz);

    for (size_t i = 0; i < sz; ++i) {
        deque_push_back(&r, sz - i);
    }

    deque _d;
    deque_create_fill(&_d, sz, 8);

    deque c = deque_concatenate(&d, &_d);

    printf("Concatenate deques illustration:\n");
    deque_print(&c, int64_t_to_str);
    printf("\n");

    printf("Hoar\'s sort illustration:\n");
    deque_print(&r, int64_t_to_str);
    printf("\n");

    deque f = hoars_qsort_conc(r, int64_t_less, int64_t_nless);

    deque_print(&f, int64_t_to_str);
    printf("\n");


    return 0;
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ make deque_o
gcc -std=c99 -Wall -Wextra -c deque.c
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ make
gcc -std=c99 -Wall -Wextra   -c -o main.o main.c
gcc -std=c99 -Wall -Wextra -o main main.o deque.o
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ ls
deque.c  deque.o        l26-2012.djvu  main    main.o
deque.h  l25-2012.djvu  logs           main.c  makefile
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ make clean
rm -f *.o main
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ ls
deque.c  deque.h  l25-2012.djvu  l26-2012.djvu  logs  main.c  makefile
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ make
gcc -std=c99 -Wall -Wextra   -c -o main.o main.c
gcc -std=c99 -Wall -Wextra   -c -o deque.o deque.c
gcc -std=c99 -Wall -Wextra -o main main.o deque.o
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ ls
deque.c  deque.o        l26-2012.djvu  main    main.o
deque.h  l25-2012.djvu  logs           main.c  makefile
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l25n26$ ./main
Push back illustration:
{ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 }
Insert illustration:
{ 0, 1, 4, 9, 16, -10, 25, 36, 49, 64, 81 }
Pop back illustration:
{ 0, 1, 4, 9, 16, -10, 25, 36, 49, 64 }
{ 0, 1, 4, 9, 16, -10, 25, 36, 49 }
{ 0, 1, 4, 9, 16, -10, 25, 36 }
{ 0, 1, 4, 9, 16, -10, 25 }
{ 0, 1, 4, 9, 16, -10 }
{ 0, 1, 4, 9, 16 }
{ 0, 1, 4, 9 }
{ 0, 1, 4 }
{ 0, 1 }
{ 0 }
{}
Construct from array illustration:
{ 1, 2, 3, 4, 5 }
Fill deque illustration:
{ 9, 9, 9, 9, 9 }
Erase element from deque illustration:
{ 9, 9, 9, 8, 9 }
{ 9, 9, 9, 9 }
Concatenate deques illustration:
{ 9, 9, 9, 9, 8, 8, 8, 8, 8 }
Hoar's sort illustration:
{ 5, 4, 3, 2, 1 }
{ 1, 2, 3, 4, 5 }
```

9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. **Замечания автора** по существу работы: _____

11. **Выводы:** в ходе этой лабораторной работы я получил опыт работы с утилитой make и реализации некоторый линейных структур. _____

12. Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента _____