Державний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

**КУРСОВА РОБОТА**

з дисципліни «Алгоритмізація та програмування»

Тема «Програмування динамічної структури даних – черга»

Студента (ки) \_\_1\_\_ курсу АІ-212 групи

Спеціальності 122 – «Комп’ютерні науки»

\_\_\_\_\_\_\_\_\_Прокопа А. С.\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент, к.т.н. Манікаєва О.С.

\_\_\_\_\_\_\_\_\_\_Стажер-помічник\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали

м. Одеса – 2022 рік

Державний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ

студенту Прокопу Артему Сергiйовичу група АІ-212

1. Тема роботи

«Програмування динамічної структури даних – черга»

2. Термін здачі студентом закінченої роботи 29.06.2022

3. Початкові дані до проекту (роботи)

Варіант 17

Предметна область – ювелірний магазин. Реалізувати динамічну структуру даних (черга), що містить наступну інформацію: Структура: найменування виробу, вид дорогоцінного металу, вага дорогоцінного металу, кількість дорогоцінних каменів, назва каменю, вага в каратах, вартість, дата надходження в магазин. Програма повинна виконувати: додавання елемента; видалення елемента; можливість коригування даних; виведення всіх даних; формування списку елементів з вартістю виробів у заданому діапазоні; пошук виробів по введеному опису (вид металу, назва каменю); підрахунок сумарної вартості всіх виробів по введеному найменуванню; виведення всіх виробів з однаковою датою надходження; сортування по полю вартість.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)

Вступ. Теоретичні відомості про чергу. Програмна реалізація черги. Інструкція користувача. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Блок-схема алгоритму – 1 аркуш формату А1.

Завдання видано 21.03.22 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Завдання прийнято до виконання 21.03.22 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис студента)

КОД ПРОГРАМИ

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "entity-queue.h"

#include "util-process-input.h"

#include "util-ui.h"

int main() {

printf("Coursework «Jewelry Store»");

// Create empty queue

struct Jewel\_Queue jewels\_queue;

struct Jewel\_Queue\* jewels = &jewels\_queue;

init\_queue(jewels);

int exit = 0;

do {

exit = process\_input(jewels, open\_menu());

} while (!exit);

printf("\n\_\_\_\_\_\_\_\_\nProgram finished successfully\n\_\_\_\_\_\_\_\_\n");

return 0;

}

**entity-jewel.c**

#include "entity-jewel.h"

#include <stdio.h>

#include <stdlib.h>

struct Jewel\* create\_jewel() {

struct Jewel\* p\_jewel = malloc(sizeof(struct Jewel));

return p\_jewel;

}

void input\_jewel(struct Jewel\* p\_jewel) {

printf("Input jewel data by template (in one line, separated by spaces):\n");

printf("name type weight number\_of\_stones stone\_name carat\_weight price date:\n");

scanf("%s", p\_jewel->name);

scanf("%s", p\_jewel->type);

scanf("%lf", &p\_jewel->weight);

scanf("%d", &p\_jewel->number);

scanf("%s", p\_jewel->stone\_name);

scanf("%lf", &p\_jewel->carat\_weight);

scanf("%lf", &p\_jewel->price);

scanf("%s", p\_jewel->date);

}

void print\_jewel(struct Jewel\* p\_jewel) {

const char\* template =

"Name: %s\n\

Type: %s\n\

Weight: %.2lf\n\

Number of stones: %d\n\

Stone name: %s\n\

Carat weight: %.2lf\n\

Price: %.3lf\n\

Date: %s\n\_\_\_\_\_\n";

printf(

template,

p\_jewel->name,

p\_jewel->type,

p\_jewel->weight,

p\_jewel->number,

p\_jewel->stone\_name,

p\_jewel->carat\_weight,

p\_jewel->price,

p\_jewel->date);

}

**entity-jewel.h**

#ifndef ENTITY\_JEWEL

#define ENTITY\_JEWEL

struct Jewel {

char name[50];

char type[50];

double weight;

int number;

char stone\_name[50];

double carat\_weight;

double price;

char date[11];

struct Jewel\* next;

};

extern struct Jewel\* create\_jewel();

extern void input\_jewel(struct Jewel\* jewel);

extern void print\_jewel(struct Jewel\* jewel);

#endif

**subtask-add.c**

#include "entity-queue.h"

void add\_jewel(struct Jewel\_Queue\* jewels) {

struct Jewel\* p\_jewel = create\_jewel();

input\_jewel(p\_jewel);

add\_to\_queue(jewels, p\_jewel);

}

**subtask-add.h**

#include "entity-queue.h"

extern void add\_jewel(struct Jewel\_Queue\* jewels);

**subtask-price-name.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "entity-queue.h"

void calc\_price\_by\_name(struct Jewel\_Queue\* queue) {

char name[50];

printf("Input name: ");

scanf("%s", &name);

struct Jewel\* p\_jewel = queue->first;

double total\_price = 0;

if (p\_jewel != NULL) {

while (p\_jewel != NULL) {

if (!strcmp(p\_jewel->name, name)) {

total\_price += p\_jewel->price;

}

p\_jewel = p\_jewel->next;

}

}

if (total\_price) {

printf("Total price is %.2lf\n", total\_price);

} else {

printf("Total price is 0.00 (no jewels found)\n");

}

}

**subtask-price-name.h**

#include "entity-queue.h"

extern void calc\_price\_by\_name(struct Jewel\_Queue\* queue);

**subtask-search-criteria.c**

#include <stdio.h>

#include <stdlib.h>

#include "entity-queue.h"

#include "util-process-input.h"

#include "util-ui.h"

int search\_by\_criteria(struct Jewel\_Queue\* queue) {

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel == NULL) {

printf("Jewel queue is empty\n");

return 1;

} else {

const int field = open\_fields\_menu("choose field to search by");

char string[101];

int number;

const int error = input\_field\_data(field, string, &number);

if (error) {

printf("Invalid input\n");

return 1;

}

int printed = 0;

while (p\_jewel != NULL) {

if (field\_equal\_to(p\_jewel, field, string, number)) {

print\_jewel(p\_jewel);

printed = 1;

}

p\_jewel = p\_jewel->next;

}

if (!printed) {

printf("No jewels found\n");

}

return 0;

}

}

**subtask-search-criteria.h**

#include "entity-queue.h"

extern int search\_by\_criteria(struct Jewel\_Queue\* queue);

**util-process-input.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "entity-queue.h"

#include "subtask-add.h"

#include "subtask-edit-existing.h"

#include "subtask-price-name.h"

#include "subtask-price-range.h"

#include "subtask-same-date.h"

#include "subtask-search-criteria.h"

#include "subtask-sort-price.h"

#include "util-ui.h"

int field\_equal\_to(struct Jewel\* jewel, int field, char\* string, int number) {

switch (field) {

case 1: {

return !strcmp(jewel->name, string);

}

case 2: {

return !strcmp(jewel->type, string);

}

case 3: {

return jewel->weight == number;

}

case 4: {

return jewel->number == number;

}

case 5: {

return !strcmp(jewel->stone\_name, string);

}

case 6: {

return jewel->carat\_weight == number;

}

case 7: {

return jewel->price == number;

}

case 8: {

return !strcmp(jewel->date, string);

}

}

}

int input\_field\_data(int field, char\* string, int\* number) {

int success = 1;

printf("To perform search, input ");

switch (field) {

case 1: {

printf("name: ");

scanf("%s", string);

break;

}

case 2: {

printf("type: ");

scanf("%s", string);

break;

}

case 3: {

printf("weight: ");

success = scanf("%lf", number) && success;

break;

}

case 4: {

printf("number of stones: ");

success = scanf("%d", number) && success;

break;

}

case 5: {

printf("stone name: ");

scanf("%s", string);

break;

}

case 6: {

printf("carat weight: ");

success = scanf("%lf", number) && success;

break;

}

case 7: {

printf("price: ");

success = scanf("%lf", number) && success;

break;

}

case 8: {

printf("date: ");

scanf("%s", string);

break;

}

}

return !success;

}

int process\_field\_input(struct Jewel\* jewel, int input) {

int success = 1;

printf("Input new ");

switch (input) {

case 1: {

printf("name: ");

success = scanf("%s", jewel->name) && success;

break;

}

case 2: {

printf("type: ");

success = scanf("%s", jewel->type) && success;

break;

}

case 3: {

printf("weight: ");

success = scanf("%lf", &jewel->weight) && success;

break;

}

case 4: {

printf("number of stones: ");

success = scanf("%d", &jewel->number) && success;

break;

}

case 5: {

printf("stone name: ");

success = scanf("%s", jewel->stone\_name) && success;

break;

}

case 6: {

printf("carat weight: ");

success = scanf("%lf", &jewel->carat\_weight) && success;

break;

}

case 7: {

printf("price: ");

success = scanf("%lf", &jewel->price) && success;

break;

}

case 8: {

printf("date: ");

success = scanf("%s", jewel->date) && success;

break;

}

}

return !success;

}

int process\_input(struct Jewel\_Queue\* jewels, int input) {

int error = 0;

switch (input) {

case 1: {

add\_jewel(jewels);

break;

}

case 2: {

remove\_first\_from\_queue(jewels);

break;

}

case 3: {

error = edit\_existing(jewels);

break;

}

case 4: {

print\_queue(jewels);

break;

}

case 5: {

print\_queue\_in\_price\_range(jewels);

break;

}

case 6: {

search\_by\_criteria(jewels);

break;

}

case 7: {

calc\_price\_by\_name(jewels);

break;

}

case 8: {

print\_jewels\_with\_same\_date(jewels);

break;

}

case 9: {

sort\_by\_price(jewels);

break;

}

default: {

error = 1;

}

}

pause();

printf("Task finished with exit code %d\n", error);

return error;

}

**util-process-input.h**

#include "entity-queue.h"

int process\_input(struct Jewel\_Queue\* jewels, int input);

int process\_field\_input(struct Jewel\* jewel, int input);

int field\_equal\_to(struct Jewel\* jewel, int field, char\* string, int number);

int input\_field\_data(int field, char\* string, int\* number);

**subtask-price-range.c**

#include <stdio.h>

#include <stdlib.h>

#include "entity-queue.h"

void print\_queue\_in\_price\_range(struct Jewel\_Queue\* queue) {

double min\_price;

printf("Input minimal price: ");

scanf("%lf", &min\_price);

double max\_price;

printf("Input maximal price: ");

scanf("%lf", &max\_price);

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel == NULL) {

printf("Jewel queue is empty\n");

} else {

int printed = 0;

while (p\_jewel != NULL) {

if (p\_jewel->price >= min\_price && p\_jewel->price <= max\_price) {

print\_jewel(p\_jewel);

printed = 1;

}

p\_jewel = p\_jewel->next;

}

if (!printed) {

printf("No jewels found\n");

}

}

}

**subtask-price-range.h**

#include "entity-queue.h"

extern void print\_queue\_in\_price\_range(struct Jewel\_Queue\* queue);

**util-ui.c**

#include "util-ui.h"

#include <stdio.h>

#include <stdlib.h>

void pause() {

printf(">> Press RETURN to continue <<");

getchar();

getchar(); // Burn in hell whoever invented all these crappy i/o functions in C!

}

void clear() {

printf("\033[H\033[J");

}

int open\_fields\_menu(char\* action) {

clear();

printf(

"1. Name\n\

2. Type\n\

3. Weight\n\

4. Number of stones\n\

5. Stone name\n\

6. Carat weight\n\

7. Price\n\

8. Date\n\

Input number to %s → ", action);

int selected;

int is\_number = scanf("%d", &selected);

if (!is\_number || selected < 1 || selected > 8) {

printf("\n\nError: input must be an integer [1-8]. Exiting..\n");

return 10;

}

return selected;

}

int open\_menu() {

clear();

printf(

"1. Add jewel to queue\n\

2. Remove first jewel from queue\n\

3. Edit existing jewel\n\

4. Show all jewels\n\

5. Search jewels by price range\n\

6. Search jewels by criteria\n\

7. Calculate total price by jewel name\n\

8. Find all with same date\n\

9. Sort by price\n\

10. Quit\n\

Input number to run task → ");

int selected;

int is\_number = scanf("%d", &selected);

if (!is\_number || selected < 1 || selected > 10) {

printf("\n\nError: input must be an integer [1-10]. Exiting..\n");

return 10;

}

return selected;

}

**util-ui.h**

extern void pause();

extern void clear();

extern int open\_menu();

extern int open\_fields\_menu(char\* action);

**entity-queue.c**

#include "entity-queue.h"

#include <stdio.h>

#include <stdlib.h>

void init\_queue(struct Jewel\_Queue\* queue) {

queue->first = NULL;

queue->last = NULL;

}

void add\_to\_queue(struct Jewel\_Queue\* queue, struct Jewel\* p\_jewel) {

p\_jewel->next = NULL;

if (queue->first == NULL) {

queue->first = p\_jewel;

} else {

queue->last->next = p\_jewel;

}

queue->last = p\_jewel;

}

int remove\_first\_from\_queue(struct Jewel\_Queue\* queue) {

if (queue->first == NULL) {

return 0;

} else {

struct Jewel\* current\_first = queue->first;

queue->first = queue->first->next;

free(current\_first);

return 1;

}

}

void print\_queue(struct Jewel\_Queue\* queue) {

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel == NULL) {

printf("Jewel queue is empty\n");

} else {

while (p\_jewel != NULL) {

print\_jewel(p\_jewel);

p\_jewel = p\_jewel->next;

}

}

}

**entity-queue.h**

#ifndef ENTITY\_QUEUE

#define ENTITY\_QUEUE

#include "entity-jewel.h"

struct Jewel\_Queue {

struct Jewel\* first;

struct Jewel\* last;

};

extern void init\_queue(struct Jewel\_Queue\* queue);

extern void add\_to\_queue(struct Jewel\_Queue\* queue, struct Jewel\* p\_jewel);

extern int remove\_first\_from\_queue(struct Jewel\_Queue\* queue);

extern void print\_queue(struct Jewel\_Queue\* queue);

#endif

**subtask-edit-existing.c**

#include <stdio.h>

#include <stdlib.h>

#include "entity-queue.h"

#include "util-process-input.h"

#include "util-ui.h"

int edit\_existing(struct Jewel\_Queue\* queue) {

int number;

printf("Input number of jewel to edit: ");

const int valid = scanf("%d", &number);

if (!valid) {

printf("Invalid input\n");

return 1;

}

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel == NULL) {

printf("Jewel not found\n");

return 1;

} else {

for (int i = 0; i < number; ++i) {

if (i == number - 1) {

const int field = open\_fields\_menu("choose filed to edit");

const int error = process\_field\_input(p\_jewel, field);

return error;

}

p\_jewel = p\_jewel->next;

if (p\_jewel == NULL) {

printf("Jewel not found\n");

return 0;

}

}

}

}

**subtask-edit-existing.h**

#include "entity-queue.h"

extern int edit\_existing(struct Jewel\_Queue\* queue);

**subtask-sort-price.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "entity-queue.h"

void assign(struct Jewel data, struct Jewel\* jewel) {

strcpy(jewel->name, data.name);

strcpy(jewel->type, data.type);

jewel->weight = data.weight;

strcpy(jewel->stone\_name, data.stone\_name);

jewel->carat\_weight = data.carat\_weight;

jewel->price = data.price;

strcpy(jewel->date, data.date);

}

void sort\_by\_price(struct Jewel\_Queue\* queue) {

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel != NULL) {

int exit;

do {

exit = 1;

p\_jewel = queue->first;

while (p\_jewel != NULL && p\_jewel->next != NULL) {

if (p\_jewel->price > p\_jewel->next->price) {

exit = 0;

struct Jewel temp = \*p\_jewel;

assign(\*(p\_jewel->next), p\_jewel);

assign(temp, p\_jewel->next);

}

p\_jewel = p\_jewel->next;

}

} while (!exit);

}

}

**subtask-sort-price.h**

#include "entity-queue.h"

extern void sort\_by\_price(struct Jewel\_Queue\* queue);

**subtask-same-date.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "entity-queue.h"

void print\_jewels\_with\_same\_date(struct Jewel\_Queue\* queue) {

struct Jewel\* p\_jewel = queue->first;

if (p\_jewel == NULL) {

printf("Jewel queue is empty\n");

} else {

int printed = 0;

while (p\_jewel != NULL) {

struct Jewel\* p\_jewel\_2 = queue->first;

while (p\_jewel\_2 != NULL) {

if (p\_jewel\_2 != p\_jewel && !strcmp(p\_jewel\_2->date, p\_jewel->date)) {

print\_jewel(p\_jewel\_2);

printed = 1;

}

p\_jewel\_2 = p\_jewel\_2->next;

}

p\_jewel = p\_jewel->next;

}

if (!printed) {

printf("No jewels with same date found\n");

}

}

}

**subtask-same-date.h**

#include "entity-queue.h"

void print\_jewels\_with\_same\_date(struct Jewel\_Queue\* queue);

ВСТУП

**Черга** (англ. Queue) — абстрактна динамічна структура даних, для якої визначенi операцiï: додавання елементу в кiнець, видалення елементу з початку, виведення з початку до кiнця.

ПРОГРАМНА РЕАЛIЗАЦIЯ ЧЕРГИ

Програмна реалізація черги полягає у створенні динамічної структури, що містить у собі посилання на перший і останній елемент черги, а також набору вищезгаданих операцій. Кожен елемент черги повинен мати посилання наступний елемент, таким чином визначаючи зв'язок між елементами черги.

IНСТРУКЦIЯ КОРИСТУВАЧА

Щоб запустити програму, необхідно встановити компілятор GCC і скомпілювати всі файли проекту (див. розділ код програми), потім запустити бінарний файл main.

Для навігації програма надає CLI з меню вибору. Щоб вибрати пункт меню, введіть його номер і дотримуйтеся вказівок на екрані: введення даних або натискання Enter для продовження. Для завершення програми використовується відповідний пункт меню.

СКРИНШОТИ РЕЗУЛЬТАТІВ РОБОТИ





























