

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 AI-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

<u> </u> (підпис)	<u> </u> (прізвище та ініціали)
<u> </u> (підпис)	<u> </u> (прізвище та ініціали)
<u> </u> (підпис)	<u> </u> (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 Сергійовичу

група AI-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)

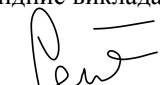
Вступ. Теоретичні відомості про чергу. Програмна реалізація черги. Інструкція користувача. Висновки.

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

Блок-схема алгоритму – 1 аркуш формату A1.

Завдання прийнято до виконання 21.03.22

(підпис викладача)



(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 (1 != 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 field 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;
        }
    }
}

```



```

    }
    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 визначені операції: додавання елемента в кінець, видалення елемента з початку, виведення з початку до кінця.

ПРОГРАМНА РЕАЛІЗАЦІЯ 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.

ІНСТРУКЦІЯ 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

```

1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task →

```

```

1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 1
Input jewel data by template (in one line, separated by spaces):
name type weight number_of_stones stone_name carat_weight price date:
Test ring 1.5 10 test 6.7 1500 2021-01-12
>> Press RETURN to continue <<

```

```

1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 4
Name: Test
Type: ring
Weight: 1.50
Number of stones: 10
Stone name: test
Carat weight: 6.70
Price: 1500.000
Date: 2021-01-12
-----
>> Press RETURN to continue <<

```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 2
>> Press RETURN to continue <<|
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 4
Jewel queue is empty
>> Press RETURN to continue <<
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 1
Input jewel data by template (in one line, separated by spaces):
name type weight number_of_stones stone_name carat_weight price date:
1 2 3 4 5 6 7 8
>> Press RETURN to continue <<
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 3
Input number of jewel to edit: 1
```

```
1. Name
2. Type
3. Weight
4. Number of stones
5. Stone name
6. Carat weight
7. Price
8. Date
Input number to choose filed to edit → 1
Input new name: test2
>> Press RETURN to continue <<
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 4
Name: test2
Type: 2
Weight: 3.00
Number of stones: 4
Stone name: 5
Carat weight: 6.00
Price: 7.000
Date: 8
-----
>> Press RETURN to continue <<
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 5
Input minimal price: 0
Input maximal price: 5
No jewels found
>> Press RETURN to continue <<
```

```
1. Name
2. Type
3. Weight
4. Number of stones
5. Stone name
6. Carat weight
7. Price
8. Date
Input number to choose field to search by → 1
To perform search, input name: 1
```

```
1. Name
2. Type
3. Weight
4. Number of stones
5. Stone name
6. Carat weight
7. Price
8. Date
Input number to choose field to search by → 1
To perform search, input name: 1
No jewels found
>> Press RETURN to continue <<
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 7
Input name: test2
Total price is 7.00
>> Press RETURN to continue <<|
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 8
No jewels with same date found
>> Press RETURN to continue <<
```



```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 9
>> Press RETURN to continue <<|
```

```
1. Add jewel to queue
2. Remove first jewel from queue
3. Edit existing jewel
4. Show all jewels
5. Search jewels by price range
6. Search jewels by criteria
7. Calculate total price by jewel name
8. Find all with same date
9. Sort by price
10. Quit
Input number to run task → 10
>> Press RETURN to continue <<
Task finished with exit code 1

-----
Program finished successfully
-----

Press any key to continue...
```