Лабораторная работа № 13

Цель работы

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование стандартных обобщенных алгоритмов из библиотеки STL в ОО программе

Постановка задачи

Задача 1.

1. Создать последовательный контейнер - двунаправленная очередь
2. Заполнить его элементами пользовательского типа (Money). Для пользовательского типа перегрузить необходимые операции.
3. Найти максимальный элемент и добавить его в конец контейнера (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill()).
4. Найти элемент с заданным ключом и удалить его из контейнера (использовать алгоритмы remove(),remove_if(), remove_copy_if(),remove_copy())
5. К каждому элементу добавить среднее арифметическое элементов контейнера.
6. Найти в контейнере заданный элемент (использовать алгоритмы find(), find_if(), count(), count_if()).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each()) .
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2.

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (Вектор). Для пользовательского типа перегрузить необходимые операции.
3. Найти максимальный элемент и добавить его в конец контейнера (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill()).
4. Найти элемент с заданным ключом и удалить его из контейнера (использовать алгоритмы remove(),remove_if(), remove_copy_if(),remove_copy())
5. К каждому элементу добавить среднее арифметическое элементов контейнера.
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find_if(), count(), count_if()).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each()) .
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3

1. Создать ассоциативный контейнер - множество.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Найти максимальный элемент и добавить его в конец контейнера (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill()).

4. Найти элемент с заданным ключом и удалить его из контейнера (использовать алгоритмы remove(),remove_if(), remove_copy_if(),remove_copy())
5. К каждому элементу добавить среднее арифметическое элементов контейнера.
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find_if(), count(), count_if()).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each()) .
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Описание класса

Для выполнения работы использовался класс Money, который представляет денежную сумму в рублях и копейках:

```
class Money {
    long rubles;
    int kopecks;

public:
    // Конструкторы
    Money();
    Money(long r, int k);
    Money(const Money& m);

    // Операторы сравнения
    bool operator<(const Money& other) const;
    bool operator==(const Money& other) const;

    // Арифметические операции
    Money operator+(const Money& other) const;
    Money operator/(int divisor) const;

    // Ввод/вывод
    friend istream& operator>>(istream& in, Money& m);
    friend ostream& operator<<(ostream& out, const Money& m);
};
```

Определение компонентных функций

Реализация класса Money

```
Money::Money() : rubles(0), kopecks(0) {}

Money::Money(long r, int k) : rubles(r), kopecks(k) {
    // Нормализация копеек
    if (kopecks >= 100 || kopecks < 0) {
        rubles += kopecks / 100;
        kopecks %= 100;
    }
    if (kopecks < 0) {
```

```
        rubles--;
        kopecks += 100;
    }
}

bool Money::operator<(const Money& other) const {
    if (rubles != other.rubles)
        return rubles < other.rubles;
    return kopecks < other.kopecks;
}

Money Money::operator+(const Money& other) const {
    long totalRub = rubles + other.rubles;
    int totalKop = kopecks + other.kopecks;
    return Money(totalRub, totalKop);
}

istream& operator>>(istream& in, Money& m) {
    cout << "Enter rubles: ";
    in >> m.rubles;
    cout << "Enter kopecks: ";
    in >> m.kopecks;
    // Нормализация при вводе
    if (m.kopecks >= 100 || m.kopecks < 0) {
        m.rubles += m.kopecks / 100;
        m.kopecks %= 100;
    }
    return in;
}
```

Определение глобальных функций

Для работы с deque

```
void fillDeque(deque<Money>& dq) {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
        Money m;
        cin >> m;
        dq.push_back(m);
    }
}

void addMaxToEnd(deque<Money>& dq) {
    if (dq.empty()) return;

    auto maxIt = max_element(dq.begin(), dq.end());
    dq.push_back(*maxIt);
}
```

```
void removeByValue(deque<Money>& dq, const Money& key) {
    dq.erase(remove(dq.begin(), dq.end(), key), dq.end());
}

void addAverage(deque<Money>& dq) {
    if (dq.empty()) return;

    Money sum = accumulate(dq.begin(), dq.end(), Money());
    Money avg = sum / dq.size();

    transform(dq.begin(), dq.end(), dq.begin(),
        [avg](Money m) { return m + avg; });
}
```

Для работы со stack

```
void processStack(stack<Money>& s) {
    if (s.empty()) return;

    // Находим максимальный элемент
    vector<Money> temp;
    while (!s.empty()) {
        temp.push_back(s.top());
        s.pop();
    }

    auto maxIt = max_element(temp.begin(), temp.end());
    temp.push_back(*maxIt);

    // Восстанавливаем стек
    for (auto it = temp.rbegin(); it != temp.rend(); ++it) {
        s.push(*it);
    }
}
```

Функция main()

```
int main() {
    // Задача 1 - работа с deque<Money>
    deque<Money> moneyDeque;
    fillDeque(moneyDeque);

    cout << "\nOriginal deque:";
    for_each(moneyDeque.begin(), moneyDeque.end(),
        [](const Money& m) { cout << m << " "; });

    addMaxToEnd(moneyDeque);
    cout << "\nAfter adding max:";
```

```
    for_each(moneyDeque.begin(), moneyDeque.end(),
        [](const Money& m) { cout << m << " "; });

    Money key;
    cout << "\nEnter money to remove: ";
    cin >> key;
    removeByValue(moneyDeque, key);

    addAverage(moneyDeque);
    cout << "\nAfter adding average:";
    for_each(moneyDeque.begin(), moneyDeque.end(),
        [](const Money& m) { cout << m << " "; });

    // Задача 2 - работа со stack<Money>
    stack<Money> moneyStack;
    // ... заполнение стека
    processStack(moneyStack);

    return 0;
}
```

Объяснение результатов работы программы

Программа демонстрирует использование стандартных алгоритмов STL для работы с контейнерами:

- Для deque:
  - Использование max_element для поиска максимального элемента
  - Применение remove для удаления элементов по значению
  - Использование accumulate и transform для добавления среднего значения
  - Алгоритм for_each для вывода элементов
- Для stack:
  - Поскольку stack - адаптер контейнера, для работы с алгоритмами STL необходимо временно копировать элементы в другой контейнер (например, vector)
  - Демонстрация ограничений адаптеров контейнеров при работе с алгоритмами STL
- Особенности работы с пользовательским типом:
  - Необходимость перегрузки операторов сравнения для работы с алгоритмами (<, ==)
  - Реализация арифметических операций для выполнения вычислений
  - Перегрузка операторов ввода/вывода для удобной работы Программа успешно выполняет все поставленные задачи, демонстрируя эффективное использование алгоритмов STL с пользовательскими типами данных в различных видах контейнеров.