Вопросы к экзамену по модулю ООП. Зима 2024-25

1. Язык. Алфавит. Классификации языков программирования по уровню, по типизации, по типу трансляции.
2. Язык. Алфавит. Классификации языков программирования по подходу.
3. Объектно-ориентированное программирование. Основные определения. Виды отношений между классами. Примеры.
4. Лексика языков C/C++. Идентификаторы, литералы, разделители, операторы, ключевые слова. Примеры.
5. Синтаксис языков C/C++. Унарные, бинарные, тернарные и сложные операторы (условий, выбора, циклов). Примеры.
6. Синтаксис языков C/C++. Функции. Примеры.
7. Синтаксис языков C/C++. Классы. Примеры.
8. Статическая и динамическая память. Работа с динамической памятью в языке C.
9. Статическая и динамическая память. Работа с динамической памятью в языке C++.
10. Массивы в языках C/C++. Принципы хранения и работы.
11. Система ввода/вывода в языке C. Стандартные потоки ввода. Файлы. Функции взаимодействия с потоками ввода. Примеры.
12. Система ввода/вывода в языке C++. Стандартные потоки ввода. Файлы. Функции, классы и объекты взаимодействия с потоками ввода. Примеры.
13. Строки в языке C. Функции работы со строками. Примеры.
14. Строки в языке C++. Функции, классы и объекты работы со строками. Примеры.
15. Создание классов в языке C++. Методы инкапсуляции. Конструкторы и деструкторы классов в языке C++. Примеры.
16. Создание классов в языке C++. Перегрузка операторов в языке C++. Дружественные операторы. Примеры.
17. Шаблонизирование функций и классов.
18. Связные списки. Хранение. Реализация. Примеры.
19. Коллекции C++. Итераторы. Виды и принцип работы.
20. Коллекции C++. array. vector. string. Принцип работы.
21. Коллекции C++. list. forward\_list. deque. Принцип работы.
22. Функциональное программирование. Функтор. Предикат. Примеры.
23. Функциональное программирование. Функция как параметр функции.
24. Функциональное программирование. Анонимные функции.
25. Функциональное программирование. Неограниченное количество параметров функции
26. Создание классов в языке C++. Наследование. Инкапсуляция. Виртуализация. Примеры.
27. Коллекции C++. set. map. unordered\_set. unordered\_map deque. Принципы работы.
28. Параллельное программирование. Примеры.
29. QT. Принципы работы. Наблюдатель. Запуск приложения.
30. QT. Компоненты.
31. QT. Сигнально-слотовая модель.
32. QT. Базовые события.

**1. Язык. Алфавит. Классификации языков программирования по уровню, по типизации, по типу трансляции.**

Язык – совокупность алфавита, лексики, синтаксиса, семантики и прагматики, предназначенная для кодирования информации.

Алфавит - это набор символов, используемых для написания программ.

Классификации языков программирования:

\* По уровню: низкоуровневые (ассемблер), высокоуровневые (C, C++, Java), сверхвысокие (Python, Ruby)

\* По типизации: статически типизированные – напрямую обозначение типа данных (C, C++), динамически типизированные – определение типа данных по присвоенному значению (Python, JavaScript)

\* По типу трансляции: интерпретируемые (Python, JavaScript), компилируемые (C, C++, Java)

**2. Язык. Алфавит. Классификации языков программирования по подходу.**

Язык – совокупность алфавита, лексики, синтаксиса, семантики и прагматики, предназначенная для кодирования информации.

Алфавит - это набор символов, используемых для написания программ.

Классификации языков программирования по подходу:

\* Объектно-ориентированное программирование – осн единица данных объект (C++, Java)

\* Процедурное программирование – языки, где можно собрать операторы в подпрограммы или функции (C, Pascal)

\* Функциональное программирование - языки, где процессы представленны в виде матем. функций (Haskell, Lisp)

**3. Объектно-ориентированное программирование. Основные определения. Виды отношений между классами. Примеры.**

Объектно-ориентированное программирование - это подход к программированию, который фокусируется на объектах и их взаимодействиях. Основные определения:

\* Класс - это шаблон для создания объектов

\* Объект - это экземпляр класса

\* Атрибут - это переменная, принадлежащая объекту

\* Метод - это функция, принадлежащая объекту

Виды отношений между классами:

\* Реализация – один из классов реализует абстрактное поведение другого (сам он ничего не делает)

\* Наследование - когда один класс наследует атрибуты и методы другого класса

\* Ассоциация – ни у одного объекта нет выраженного хар-ра владения над другим

\* Агрегация - когда один класс владеет другим(-и) полностью

**4. Лексика языков C/C++. Идентификаторы, литералы, разделители, операторы, ключевые слова. Примеры.**

Лексика языков C/C++:

\* Идентификаторы - это имена переменных, функций и классов, которые не меняются при изменении кодировки(for, while)

\* Литералы - это лексика, описывающая значение:

1) целочисленные (0b – двоичное число, 0x – 16ричное число, 0 – 8ричное, L/LL в конце – long/long long)

2) дробные (. – дробное число(.5 = 0.5, 3. = 3.0 float), .f – одинарная точность, .d – 2-ная точность)

3) символьные(\ - показывает, что следующий символ не управляющий - \n, \r, \t, также \ используется для обозначения кода символа ‘\040’ = 32)

4) строковые(\b – не читай символ перед литералом, но и не изменяй память для хранения, \0 – игнорь, что дальше, но жри оперативку)

5) логические(true, false)

6) Нулл литералы (nullptr – описывает отсутствие значения)

\* Разделители - это символы, которые отделяют элементы программы(пробел, ->, \n, (), {}, “,”, ;)

\* Операторы - это символы, которые выполняют действия над операндами:

1) арифметические(+, -, /, %)

2) задающие(=, +=, -=, ++, --)

3) сравнительные(>, <, ==, !=)

4) логические(!, &&, ||)

5) битовые(&, |, ~, ^, >>, <<)

6) адресные(&, \*, точка , ->, [])

7) другие(тернарный оператор, if else, - условные, for, while, do while – опер. цикла, оператор вызова функции, преобразование типа данных, “,” , ; , {оператор печесиления}, sizeof)

\* Ключевые слова – заранее определённые идентификаторы, имеющие спец значение(if, for, bool, break, struct)

**5. Синтаксис языков C/C++. Унарные, бинарные, тернарные и сложные операторы (условий, выбора, циклов). Примеры.**

Синтаксис языков C/C++:

\* Унарные операторы - это операторы, которые действуют на один операнд, такие как унарный плюс и унарный минус(+5, -5)

\* Бинарные операторы - это операторы, которые действуют на два операнда, такие как сложение и умножение

\* Тернарные операторы - это операторы …? … : …

\* Сложные операторы - это операторы, которые действуют на несколько операндов, такие как операторы условия, выбора и циклов

**6. Синтаксис языков C/C++. Функции. Примеры.**

\* Функции - это блоки кода, которые можно вызывать несколько раз из разных частей программы

\* Функции могут принимать аргументы и возвращать значения

\* Функции могут быть объявлены и определены в любом месте программы

Функции в C:

void hello() // принимает сколько угодно аргументов, которые ни на что не влияют

{

    printf("Hello!\n");

}

Функции C++:

void hello() // не принимает аргументов

{

cout << “Hello!\n”;

}

**7. Синтаксис языков C/C++. Классы. Примеры.**

Синтаксис языков C/C++:

\* Классы - это шаблоны для создания объектов

\* Классы могут содержать атрибуты и методы

\* Классы могут наследовать атрибуты и методы других классов

1. Концепция классов:

- В C++ классы являются основой ООП и позволяют создавать объекты, которые объединяют данные и методы (функции), работающие с этими данными.

- В C нет классов, вместо этого используются структуры (`struct`), которые могут содержать только данные, но не методы.

2. Инкапсуляция:

- В C++ классы позволяют использовать инкапсуляцию, что означает, что можно скрывать данные и предоставлять доступ к ним только через специальные методы (геттеры и сеттеры). Это помогает защитить данные от неправильного использования.

- В C структуры не имеют встроенных механизмов инкапсуляции, и все поля доступны напрямую.

3. Наследование:

- C++ поддерживает наследование, позволяя создавать новые классы на основе существующих, что способствует повторному использованию кода.

- В C нет концепции наследования, поэтому создание иерархий классов невозможно.

4. Полиморфизм:

- C++ поддерживает полиморфизм, что позволяет использовать один интерфейс для разных типов данных (например, через виртуальные функции).

- В C полиморфизм реализуется только через функции и указатели, но не так удобно и мощно, как в C++.

**8. Статическая и динамическая память. Работа с динамической памятью в языке C.**

Статическая память - это память, которая выделена заранее и остается неизменной во время выполнения программы.

Динамическая память - это память, которая выделяется во время выполнения программы.

Динамическая память:

Память, динамически выделенная с использованием функций calloc(), malloc(), может быть освобождена с использованием функции free(указатель)

void\* malloc(РазмерМассиваВБайтах);  
void\* calloc(ЧислоЭлементов, РазмерЭлементаВБайтах);

free(x); //память, выделенная под x потчистится

int \*x = (int\*) malloc(10);

int \*y = (int\*) calloc(10, sizeof(int));

free(x);

free(y);

Статическая память – выделяется при инициализации

**9. Статическая и динамическая память. Работа с динамической памятью в языке C++.**

В языке C++ динамическая память может быть выделена с помощью оператора new и удалена с помощью оператора delete.

Для переменных:

int \*p = new int;

delete p;

Для массивов:

int \*p\_arrray = new int[1010];

delete[] p\_array;

**10. Массивы в языках C/C++. Принципы хранения и работы.**

Массивы - это коллекции элементов одного типа, хранящиеся в последовательных ячейках памяти.

Массивы в C и C++ имеют много общего, но также существуют и различные отличия, которые следует учитывать:

1. Объявление и инициализация: В C и C++ массивы могут быть объявлены и инициалированы одинаково. Пример:

int arr[5] = {1, 2, 3, 4, 5};

2. Размер массивов: В C необходимо, чтобы размер массива был известен на этапе компиляции (для статических массивов). В C++ также можно использовать стандартные контейнеры, такие как `std::vector`, которые могут изменять свой размер динамически.

3. Строки: В C строки представляют собой массивы символов, заканчивающиеся нуль-символом (`\0`). В C++ можно использовать класс `std::string`, который упрощает работу со строками и предоставляет множество встроенных методов.

4. Аргументы функции: В C и C++ массивы могут передаваться в функции как указатели. Однако в C++ можно также использовать ссылки на массивы или шаблоны для большей гибкости.

Несмотря на различия, базовое использование массивов в C и C++ остается достаточно одинаковым, так как C++ является расширением C и сохраняет его основные концепции.

**11. Система ввода/вывода в языке C. Стандартные потоки ввода. Файлы. Функции взаимодействия с потоками ввода. Примеры.**

В языке C стандартные потоки ввода и вывода реализованы через библиотеку <stdio.h>. Основные стандартные потоки включают stdin (ввод), stdout (вывод) и stderr (ошибки). Эти потоки можно использовать для работы с консолью и файлами.

1. Ввод вывод числа, которое ввёл пользователь

int number;

printf("Введите число: ");

scanf("%d", &number);

printf("Вы ввели: %d\n", number);

2. Работа с файлом

#include <stdio.h>

int main() {

FILE \*file;

char buffer[100];

// Открытие файла для записи

file = fopen("example.txt", "w");

if (file == NULL) {

perror("Ошибка при открытии файла");

return 1;

}

fprintf(file, "Привет, мир!\n");

fclose(file);

// Открытие файла для чтения

file = fopen("example.txt", "r");

if (file == NULL) {

perror("Ошибка при открытии файла");

return 1;

}

// Чтение из файла

while (fgets(buffer, sizeof(buffer), file) != NULL) {

printf("%s", buffer);

}

fclose(file);

return 0;

}

**12. Система ввода/вывода в языке C++. Стандартные потоки ввода. Файлы. Функции, классы и объекты взаимодействия с потоками ввода. Примеры.**

В C++ стандартные потоки ввода и вывода реализованы через библиотеку <iostream>. Основные стандартные потоки включают std::cin (ввод), std::cout (вывод) и std::cerr (ошибки). C++ также предоставляет возможности обработки файлов через классы в библиотеке <fstream>.

1. Ввод вывод числа, которое ввёл пользователь

int number;

std::cout << "Введите число: ";

std::cin >> number;

std::cout << "Вы ввели: " << number << std::endl;

2. Работа с файлом

#include <iostream>

#include <fstream>

#include <string>

int main() {

std::ofstream outFile("example.txt");

if (!outFile) {

std::cerr << "Ошибка при открытии файла для записи" << std::endl;

return 1;

}

outFile << "Привет, мир!" << std::endl;

outFile.close();

std::ifstream inFile("example.txt");

if (!inFile) {

std::cerr << "Ошибка при открытии файла для чтения" << std::endl;

return 1;

}

std::string line;

while (std::getline(inFile, line)) {

std::cout << line << std::endl;

}

inFile.close();

return 0;

}

**13. Строки в языке C. Функции работы со строками. Примеры.**

В языке C строки представляют собой массивы символов, которые завершаются нулевым символом (\0). Этот нулевой символ используется для обозначения конца строки.

strlen(str): Возвращает длину строки (исключая нулевой символ).

strcpy(to, from): Копирует одну строку в другую.

strcat(str\_f, str\_s): Объединяет две строки(в первую)

strcmp(str\_f, str\_s): Сравнивает две строки(0, если равны, >0, если первый несовпадающий символ в первой строке больше, иначе <0)

strchr(str, char): Находит первое вхождение символа в строку.

**14. Строки в языке C++. Функции, классы и объекты работы со строками. Примеры.**

В C++ строки — это последовательности символов, которые могут быть представлены как массивы символов или объекты класса std::string.

Базовые функции для работы со строками

Стандартная библиотека C++ предоставляет несколько функций для работы со строками, которые находятся в заголовочном файле <string>. Вот некоторые из них:

std::string: класс для работы со строками.

std::getline(): функция для чтения строки из потока.

std::stoi(): функция для преобразования строки в целое число.

std::stod(): функция для преобразования строки в число с плавающей точкой.

Базовые методы класса std::string

Класс std::string предоставляет несколько методов для работы со строками. Вот некоторые из них:

size(): возвращает длину строки.

empty(): проверяет, является ли строка пустой.

at(): возвращает символ по индексу.

find(): находит первое вхождение подстроки.

replace(): заменяет подстроку другой.

**15. Создание классов в языке C++. Методы инкапсуляции. Конструкторы и деструкторы классов в языке C++. Примеры.**

В C++ классы являются основными единицами организации кода, которые позволяют определять типы данных и поведение объектов. Давайте рассмотрим основные аспекты создания классов в C++, включая методы инкапсуляции, конструкторы и деструкторы классов, а также приведем примеры.

Методы инкапсуляции

Инкапсуляция — это механизм, позволяющий скрыть внутреннюю реализацию класса и предоставить доступ к его членам только через определенные методы.

Открытые члены(public): доступны извне класса.

Закрытые члены(private): доступны только внутри класса.

Конструкторы и деструкторы классов

Конструктор(название класса): специальный метод, который вызывается при создании объекта класса.

Деструктор(~название класса): специальный метод, который вызывается при удалении объекта класса.

**16. Создание классов в языке C++. Перегрузка операторов в языке C++. Дружественные операторы. Примеры.**

Перегрузка операторов — это механизм, позволяющий определять поведение операторов для пользовательских типов данных.

Дружественные операторы — это функции, которые не являются членами класса, но имеют доступ к его закрытым членам.

Объявление дружественного оператора: операторы, объявленные как дружественные внутри класса.

1. Перегрузка операторов + и –

class Complex {

private:

double real\_;

double image\_;

public:

Complex(double real, double imag) : real\_(real), imag\_(imag) {}

double getReal() const { return real\_; }

double getImag() const { return imag\_; }

Complex Operator+(const Complex& other) const {

return Complex(real\_ + other.real\_, imag\_ + other.imag\_);

}

Complex Operator-(const Complex& other) const {

return Complex(real\_ - other.real\_, imag\_ - other.imag\_);

}

};

2.

class B;

class A {

public:

int Func1( B& b );

private:

int Func2( B& b );

};

class B {

private:

int\_b;

friend int A::Func1( B& );

};

Класс friend — это класс, все функции-члены которого являются дружественными функциями класса, т. е. функции-члены которого имеют доступ к закрытым и защищенным членам другого класса.

**17. Шаблонизирование функций и классов.**

Шаблонизирование функций и классов - это возможность определять функции и классы, которые могут работать с разными типами данных.

1) шаблонизирование функции

template <typename El>

void f(El k) {

cout << k << std::endl;

}

2) шаблонизирование класса

template <typename T>

class MyClass {

private:

T data;

public:

MyClass(T d) : data(d) {}

T getData() const {

return data;

}

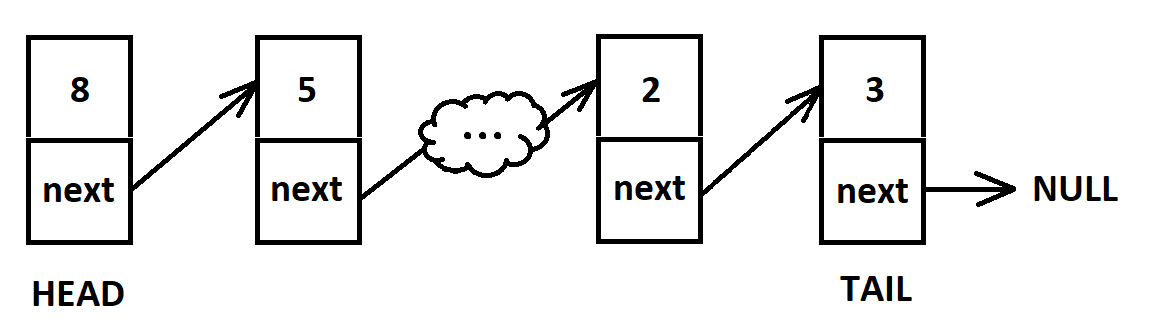
void setData(T d) {

data = d;

}

};

**18. Связные списки. Хранение. Реализация. Примеры.**



structNode {  
 int val;  
 Node\* next;  
  
 Node(int val) : val(val), next(nullptr) {}  
};

**class** List

{

**private**:

Node \*head; //"голова" связанного списка

**public**:

List() //конструктор класса без параметров

{

head = **NULL**; //первого элемента пока нет

}

//метод, добавляющий новый узел в список

**void** addNode(**int** d)

{

Node \*nd = new Node; //динамически создаем новый узел

nd->data = d; //задаем узлу данные

nd->next = **NULL**; //новый узел в конце, поэтому NULL

**if**(head == **NULL**) //если создаем первый узел

head = nd;

**else** //если узел уже не первый

{

Node \*current = head;

//ищем в цикле предшествующий последнему узел

**while**(current->next != **NULL**)

current = current->next;

//предшествующий указывает на последний

current->next = nd;

}

}

**19. Коллекции C++. Итераторы. Виды и принцип работы.**

Итераторы - это объекты, предоставляющие доступ к элементам коллекции и обеспечивающие навигацию по ним

Виды итераторов:

1. LegacyIterator – абстрактная сущность, ничего не умеет, от неё наследуются все остальные
2. LegacyOutputIterator – умеет писать в текущий элемент коллекции
3. LegacyInputIterator – умеет читать текущий элемент
4. LegacyFrowardIterator – умеет читать и перебирать элементы слева направо(через инкремент ++)
5. LegacyBidirectionalIterator – умеет читать и перепбирать элементы в любом направлении(через инкремент ++ и декремент --)
6. LedacyRandomAccessIterator – может читать и перебирать на сколько угодно элементов в любую сторону(++, --, +2, -3, +10, -1000)
7. LegacyContiguousIterator – умеет всё, кроме записи на неприрывном объекте

**20. Коллекции C++. array. vector. string. Принцип работы**

\* array – аналог стандартного массива, отличие заключается в том, что в array представлены методы(size(), at(index), front(), back(), fill(x)), которые отсутствуют в стандартном массиве C++

\* vector - это коллекция, которая хранит элементы в динамической памяти, то есть может динамически изменять свой размер по необходимости

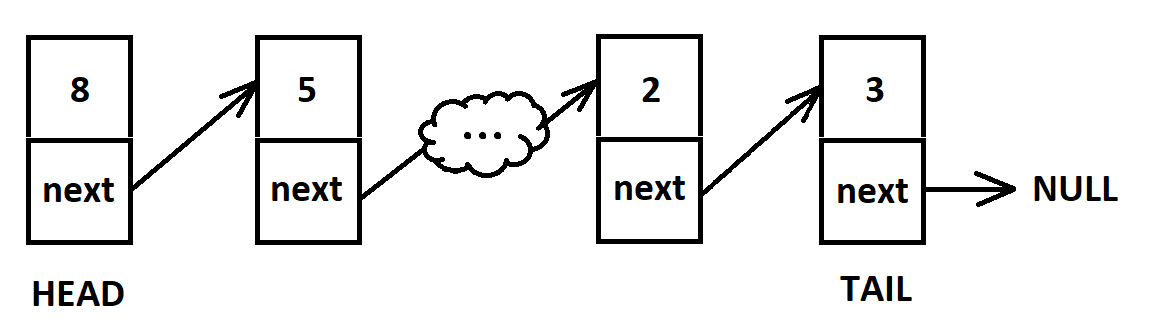
Увеличение размера вектора происходит быстро пока он меньше объёма массива. Когда нужно увеличить размер массива, а свободного места в нём нет, создаётся ещё один массив большего объёма, все элементы старого объёма копируются в новый массив, ссылка на старый массив удаляется

\* string – коллеция для работы со строковым типом данных. Внутри очень похож на vector<char>. В string представлен удобный функционал для работы со строками: объединение строк, обращение по индексу, begin(), end(), empty(), clear(), find(), insert(), erase() и другие

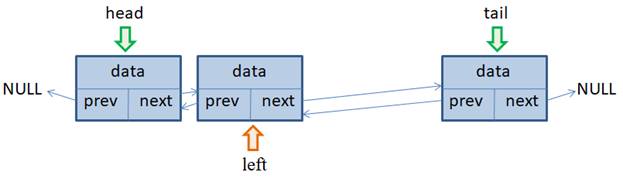
**21. Коллекции C++. list. forward\_list. deque. Принцип работы**

Коллекции C++:

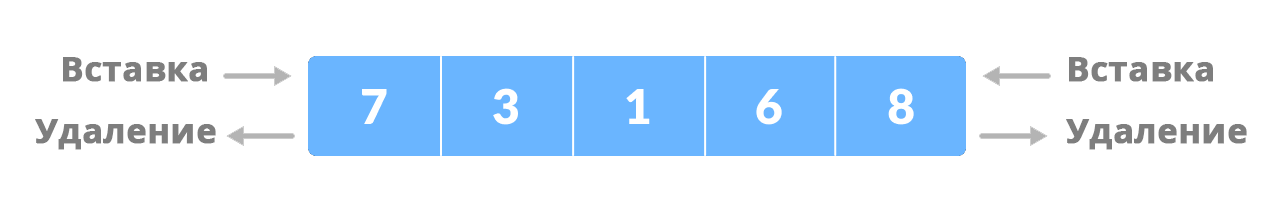
\* list - это коллекция, которая представляет собой односвязный список



\* forward\_list - это коллекция, которая представляет собой двусвязный список



\* deque - это коллекция, которая представляет собой двустороннюю очередь



**22. Функциональное программирование. Функтор. Предикат. Примеры**

Функциональное программирование - это подход к программированию, который фокусируется на функциях и их взаимодействиях.

Функтор - это объект, который может быть рассмотрен как функция.

Предикат – это частный случай функтора, возвращающий логическое значение.

Пример функтора y = kx + b:

Class F{

Private:

Double k;

Double b;

Public:

F(double k, double b): k(k), b(b){

Double calc(double x) {

Return k \* x + b;

}

Double operator()(double x){

Return calc(x);

}

};

Функтор с шаблоном:

Template<int k, int b>

Int f(int x){

Return k \* x + b;

}

global:

auto f1 = f<1, 1>

auto f2 = f<2, 3>

int main() {

std::cout << f1(2) << std::endl;

std::cout << f2(8) << std::endl;

}

**23. Функциональное программирование. Функция как параметр функции.**

Функция как параметр функции - это возможность передавать функцию в качестве аргумента другой функции.

Функции, в которые передается другая функция – функции высшего порядка

Пример:

Template <class FUP, class SUP>

Class And{

Private:

FUP f;

SUP s;

Public:

And(FUP f, SUP s): f(f), s(s){}

Bool operator()(int x){

Return f(x) && s(x);

}

};

**24. Функциональное программирование. Анонимные функции.**

Анонимные функции - это функции, которые не имеют имени и могут быть определены прямо в коде (одноразовые)

Сравнение двух чисел через анонимную(лямбда) функцию

[](int a, int b) { return a > b? a: b}

**25. Функциональное программирование. Неограниченное количество** **параметров функции**

Неограниченное количество параметров функции - это возможность определять функции, которые могут принимать любое количество аргументов.

**26. Создание классов в языке C++. Наследование. Инкапсуляция. Виртуализация. Примеры.**

Создание классов в C++ осуществляется с помощью ключевого слова `class`. Классы позволяют объединять данные и методы в одной сущности, представляющей объект. Например:

class Animal {

public:

void speak() {

std::cout << "Animal speaks" << std::endl;

}

};

Наследование позволяет создавать новый класс на основе уже существующего, тем самым продлевая его функциональность. Используются ключевые слова `public`, `protected`, `private`. Например:

class Dog : public Animal {

public:

void speak() {

std::cout << "Dog barks" << std::endl;

}

};

Здесь класс `Dog` наследует класс `Animal`. Существует множество типов наследования:

- Простое (одиночное)

- Множественное (класс может наследоваться от нескольких классов),

- Многоуровневое (классы могут наследоваться друг от друга последовательно).

Инкапсуляция заключается в скрытии внутреннего состояния объекта от внешнего доступа, что достигается с помощью модификаторов доступа: `private`, `protected`, `public`. Например:

class Encapsulated {

private:

int secretCode;

public:

void setSecretCode(int code) {

secretCode = code;

}

int getSecretCode() {

return secretCode;

}

};

\*\*Виртуализация\*\* используется для реализации полиморфизма. Виртуальные функции позволяют вызвать переопределённый метод производного класса через указатель на базовый класс. Виртуальные функции объявляются с помощью ключевого слова `virtual`. Пример:

class Base {

public:

virtual void show() {

std::cout << "Base class" << std::endl;

}

};

class Derived : public Base {

public:

void show() override {

std::cout << "Derived class" << std::endl;

}

};

void function(Base\* b) {

b->show(); // Вызов полиморфной функции

}

**27. Коллекции C++. set. map. unordered\_set. unordered\_map. deque. Принципы работы.**

C++ предлагает несколько стандартных контейнеров, таких как `set`, `map`, `unordered\_set`, `unordered\_map`, и `deque`, которые обеспечивают удобные механизмы для работы с данными.

set — это коллекция уникальных элементов, организованных в сортированном порядке с использованием дерева (обычно красно-черное дерево). Поиск, вставка и удаление элементов осуществляются за логарифмическое время. Например:

std::set<int> mySet;

mySet.insert(3);

mySet.insert(1);

mySet.insert(2);

map — это ассоциативный массив, где каждому ключу соответствует значение. Ключи также уникальны и сохраняются в отсортированном порядке. Вставка и поиск выполняются за логарифмическое время. Пример:

std::map<std::string, int> myMap;

myMap["Alice"] = 30;

myMap["Bob"] = 25;

unordered\_set и unordered\_map реализованы с использованием хеш-таблиц. Эти структуры данных обеспечивают среднее время доступа, вставки и удаления элементов за константное время, но элементы не упорядочены. Например:

std::unordered\_set<int> myUnorderedSet;

myUnorderedSet.insert(1);

myUnorderedSet.insert(2);

deque (двунаправленная очередь) позволяет эффективно добавлять и удалять элементы с обоих концов. Это более гибкая структура данных по сравнению с вектором, так как предоставляет постоянное время выполнения добавления и удаления в начале или в конце. Пример:

std::deque<int> myDeque;

myDeque.push\_back(1);

myDeque.push\_front(2);

Каждая из этих коллекций имеет свои особенности использования, и выбор конкретной структуры данных зависит от задач, которые необходимо решить.

**28. Параллельное программирование. Примеры.**

\*\*Параллельное программирование\*\* позволяет выполнять несколько вычислений одновременно, что существенно ускоряет выполнение задач, особенно на многоядерных процессорах. В C++ для реализации параллельного программирования используются потоки.

Потоки создаются с использованием библиотеки `<thread>`. Например, следующий пример демонстрирует создание потока, который выполняет функцию:

#include <iostream>

#include <thread>

void function() {

std::cout << "Hello from thread!" << std::endl;

}

int main() {

std::thread t(function); // Создание потока

t.join(); // Ожидание завершения потока

return 0;

}

Существует также возможность использовать библиотеку `<future>` для работы с результатами выполнения асинхронных задач. Это позволяет запустить задачу в отдельном потоке и вернуть результат:

#include <future>

int asyncFunction() {

return 42;

}

int main() {

std::future<int> result = std::async(std::launch::async, asyncFunction);

std::cout << "Result: " << result.get() << std::endl; // Получение результата

return 0;

}

Параллельное программирование также включает в себя работу с синхронизацией потоков с использованием mutex'ов (мьютексов) и других механизмов, таких как condition\_variable для организации взаимодействия между потоками, что позволяет избежать гонок данных.

**29. QT. Принципы работы. Наблюдатель. Запуск приложения.**

Qt — это мощная кросс-платформенная библиотека для разработки графических приложений. Она использует сигнал-слотовую модель, которая упрощает взаимодействие между объектами. Основным принципом работы Qt является объектно-ориентированный подход, где все элементы интерфейса представлены объектами.

Наблюдатель — это паттерн проектирования, реализованый в Qt через сигнально-слотовую модель. Объекты (наблюдатели) могут подписываться на сигналы объектов (субъектов), что позволяет им реагировать на изменения состояния. Пример:

class Sender : public QObject {

Q\_OBJECT

public:

void send() {

emit mySignal();

}

signals:

void mySignal();

};

class Receiver : public QObject {

Q\_OBJECT

public slots:

void onSignalReceived() {

std::cout << "Signal received!" << std::endl;

}

};

Запуск приложения в Qt начинается с создания объекта `QApplication` и определения главного окна, затем вызывается метод `exec()` для запуска цикла обработки событий. Пример:

#include <QApplication>

#include <QPushButton>

int main(int argc, char \*argv[]) {

QApplication app(argc, argv);

QPushButton button("Hello, World!");

button.show();

return app.exec(); // Запуск приложения

}

Здесь создается приложение, в котором отображается кнопка с текстом "Hello, World!".

**30. QT. Компоненты.**

В Qt имеется множество компонентов, которые позволяют создавать различные элементы интерфейса и управлять ими. Основные компоненты, с которыми работают разработчики, включают:

- QWidget: базовый класс для всех виджетов. Его используют для создания окон и различных элементов интерфейса.

- QMainWindow: специализированный класс для главных окон приложений с предустановленной работой с меню и статусными строками.

- QPushButton: кнопка, с которой могут быть связаны действия при нажатии.

- QLabel: служит для отображения текста или изображения.

- QLineEdit: предоставляет однострочное поле ввода.

- QTextEdit: многострочный текстовый блок для редактирования текста.

- QComboBox: выпадающий список для выбора элемента.

- QTableView: для отображения табличных данных.

Каждый из этих компонентов инкапсулирует свою функциональность и может взаимодействовать с другими компонентами через сигналы и слоты.

**31. QT. Сигнально-слотовая модель.**

Сигнально-слотовая модель в Qt — это механизм, позволяющий объектам выполнять взаимосвязанные действия. Сигналы отправляются, когда происходит определенное событие, и могут быть связаны с одним или несколькими слотами, которые реализуют обработку этого сигнала.

Каждый объект может излучать сигналы с помощью ключевого слова `emit`, как в следующем примере:

class Button : public QPushButton {

Q\_OBJECT

public:

Button(QWidget \*parent = nullptr) : QPushButton(parent) {

connect(this, &QPushButton::clicked, this, &Button::onClicked);

}

public slots:

void onClicked() {

std::cout << "Button was clicked!" << std::endl;

}

};

Здесь, когда кнопка нажата, отправляется сигнал `clicked`, который связывается со слотом `onClicked`, выполняющим определённое действие.

**32. QT. Базовые события.**

В \*\*Qt\*\* события представляют собой уведомления о произошедших действиях, например, нажатии клавиш, движении мыши, изменении состояния компонентов и т. д. Класс `QObject` и его производные классы обрабатывают события через метод `event()`, а специфические события могут быть обработаны с помощью переопределения методов обработки событий, таких как `mousePressEvent()`, `keyPressEvent()`, `paintEvent()` и других.

Пример обработки нажатия клавиши:

void MyWidget::keyPressEvent(QKeyEvent \*event) {

if (event->key() == Qt::Key\_Escape) {

close(); // Закрытие виджета при нажатии клавиши Esc

}

}

События могут передаваться в виде очереди, и контроль над их обработкой даётся разработчику. Можно также создать собственные события с помощью класса `QEvent` и отправлять их через `QCoreApplication::postEvent()`, что позволяет расширить функциональность приложения, учитывая различные ситуации, такие как пользовательский ввод или системные изменения.

#include <iostream>

#include <cstdarg>

void printNumbers(int count, ...) {

va\_list args;

va\_start(args, count);

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

int num = va\_arg(args, i);

std::cout << num << " ";

}

va\_end(args);

std::cout << std::endl;

}

int main() {

printNumbers(5, 1, 2, 3, 4, 5); // Вывод: 1 2 3 4 5

return 0;

}

***КОДЫ***

**Task List**

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QJsonDocument>

#include <QJsonArray>

#include <QJsonObject>

#include <QFile>

#include <QTextStream>

#include <QMainWindow>

#include <QLineEdit>

#include <QPushButton>

#include <QListWidget>

#include <QWidget>

#include <QLayout>

#include <QVBoxLayout>

#include <QInputDialog>

class MainWindow : public QMainWindow

{

Q\_OBJECT

QWidget \*\_centralWidget;

QVBoxLayout \*layout\_v;

QPushButton \*button\_add;

QPushButton \*button\_delete;

QListWidget \*task\_list;

QString get\_task\_name();

void save\_tasks\_file(const QString &file\_path);

void load\_tasks\_file(const QString &file\_path);

private slots:

void click\_add();

void click\_delete();

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

};

#endif // MAINWINDOW\_H

**Mainwindow.cpp**

#include "mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

this->resize(640, 480);

this->setMinimumSize(320, 240);

this->setWindowTitle("Task\_List");

this->\_centralWidget = new QWidget(this);

this->setCentralWidget(this->\_centralWidget);

//Ð±Ñ€Ð°Ñ‚ÑŒ ÑÑ‚Ð¸Ð»Ð¸ Ñ web-Ð¿Ñ€Ð¾Ñ„ÐµÑÑÐ¸Ð¾Ð½Ð°Ð»Ñ‹ Ð½Ð¸ÐºÑ‚Ð¾ Ð½Ðµ Ð·Ð°Ð¿Ñ€ÐµÑ‰Ð°Ð» :) R-ÑÑ‹Ñ€Ð°Ñ ÑÑ‚Ñ€Ð¾ÐºÐ°

this->\_centralWidget->setStyleSheet(R"(

QWidget {

background-color: #2c3e50; /\* Ð¢Ñ‘Ð¼Ð½Ñ‹Ð¹ Ñ„Ð¾Ð½ \*/

color: white;

}

QPushButton {

background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,

stop:0 #ff7eb3, stop:1 #ff758c); /\* Ð“Ñ€Ð°Ð´Ð¸ÐµÐ½Ñ‚ \*/

color: white;

font-size: 16px;

font-weight: bold;

border: 2px solid #ff5370;

border-radius: 10px;

padding: 10px 20px;

transition: all 0.3s ease-in-out;

}

QPushButton:hover {

background: qlineargradient(spread:pad, x1:0, y1:1, x2:1, y2:0,

stop:0 #ff5370, stop:1 #ff758c); /\* ÐžÐ±Ñ€Ð°Ñ‚Ð½Ñ‹Ð¹ Ð³Ñ€Ð°Ð´Ð¸ÐµÐ½Ñ‚ \*/

border-color: #ff2e63;

transform: scale(1.05); /\* Ð›Ñ‘Ð³ÐºÐ¾Ðµ ÑƒÐ²ÐµÐ»Ð¸Ñ‡ÐµÐ½Ð¸Ðµ \*/

}

QPushButton:pressed {

background: #ff2e63;

border-color: #d90429;

padding: 9px 19px; /\* Ð­Ñ„Ñ„ÐµÐºÑ‚ Ð½Ð°Ð¶Ð°Ñ‚Ð¸Ñ \*/

}

QPushButton:disabled {

background: #7f8c8d;

color: #bdc3c7;

border-color: #95a5a6;

}

)");

this->layout\_v = new QVBoxLayout();

this->\_centralWidget->setLayout(this->layout\_v);

this->task\_list = new QListWidget(this->\_centralWidget);

this->button\_add = new QPushButton("Add", this->\_centralWidget);

this->button\_delete = new QPushButton("Delete", this->\_centralWidget);

connect(this->button\_add, &QPushButton::clicked, this, &MainWindow::click\_add);

connect(this->button\_delete, &QPushButton::clicked, this, &MainWindow::click\_delete);

this->layout\_v->addWidget(this->task\_list);

this->layout\_v->addWidget(this->button\_add);

this->layout\_v->addWidget(this->button\_delete);

this->load\_tasks\_file("tasks.json");

}

QString MainWindow::get\_task\_name() {

bool ok;

QInputDialog \*inputDialog = new QInputDialog(this);

inputDialog->setWindowTitle("Ð’Ð²ÐµÐ´Ð¸Ñ‚Ðµ Ð½Ð°Ð·Ð²Ð°Ð½Ð¸Ðµ");

inputDialog->setLabelText("ÐÐ°Ð·Ð²Ð°Ð½Ð¸Ðµ Ð·Ð°Ð´Ð°Ñ‡Ð¸:");

inputDialog->setTextValue("");

inputDialog->setInputMode(QInputDialog::TextInput);

inputDialog->resize(400, 100);

if (inputDialog->exec() == QDialog::Accepted) {

return inputDialog->textValue();

} else {

return "";

}

}

void MainWindow::click\_add(){

qInfo() << "add";

QString name = this->get\_task\_name();

if (name.isEmpty()) return;

QListWidgetItem\* item = new QListWidgetItem(name);

this->task\_list->addItem(item);

}

void MainWindow::click\_delete(){

qInfo() << "del";

QListWidgetItem\* item = this->task\_list->currentItem();

if (item) {

delete this->task\_list->takeItem(this->task\_list->row(item));

}

}

void MainWindow::save\_tasks\_file(const QString &file\_path){

qInfo() << "save";

QJsonArray tasks\_js\_arr;

for (int i = 0; i < this->task\_list->count(); ++i) {

QListWidgetItem\* item = this->task\_list->item(i);

tasks\_js\_arr.append(item->text());

}

QJsonObject root;

root["tasks"] = tasks\_js\_arr;

QJsonDocument doc(root);

QFile file(file\_path);

if (file.open(QIODevice::WriteOnly | QIODevice::Truncate)) { // ÐŸÐµÑ€ÐµÐ·Ð°Ð¿Ð¸ÑÑŒ Ñ„Ð°Ð¹Ð»Ð°

file.write(doc.toJson(QJsonDocument::Indented));

file.close();

} else {

qInfo() << "blin";

}

}

void MainWindow::load\_tasks\_file(const QString &file\_path) {

qInfo() << "load";

QFile file(file\_path);

if (!file.exists()) {

qInfo() << "blin, lose file";

return;

}

if (!file.open(QIODevice::ReadOnly)) {

qInfo() << "blin";

return;

}

QByteArray data = file.readAll();

file.close();

QJsonDocument doc = QJsonDocument::fromJson(data);

if (!doc.isObject()) {

qInfo() << "blin, wrong in JSON";

return;

}

QJsonObject root = doc.object();

QJsonArray tasks\_js\_arr = root["tasks"].toArray();

for (const QJsonValue &val : tasks\_js\_arr) {

QString task\_name = val.toString();

if (!task\_name.isEmpty()) {

this->task\_list->addItem(task\_name);

}

}

}

MainWindow::~MainWindow() {

this->save\_tasks\_file("tasks.json");

while (this->task\_list->count() > 0) {

delete this->task\_list->takeItem(0);

}

delete this->task\_list;

delete this->button\_delete;

delete this->button\_add;

delete this->layout\_v;

delete this->\_centralWidget;

}

**Calculator**

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPushButton>

#include <QLabel>

#include <QList>

#include <QGridLayout>

#include <QWidget>

#include <QSizePolicy>

#include <QObject>

class MainWindow : public QMainWindow

{

Q\_OBJECT

QList<QPushButton\*> \*buttons;

QLabel \*display;

QWidget \*\_centralWidget;

QGridLayout \*layout;

private slots:

void on\_click(QString value);

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

};

#endif // MAINWINDOW\_H

**Mainwindow.cpp**

#include "mainwindow.h"

#include <QDebug>

#include <string>

#include <vector>

std::vector<std::string> btns\_map = {

"789/",

"456\*",

"123-",

"C0.+"

};

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

//this->resize(640, 480);

this->setWindowTitle("Calculator");

this->\_centralWidget = new QWidget(this);

this->setCentralWidget(this->\_centralWidget);

this->layout = new QGridLayout();

this->\_centralWidget->setLayout(this->layout);

this->buttons = new QList<QPushButton\*>;

this->display = new QLabel(QString("0"), this->\_centralWidget);

this->display->setStyleSheet("border: 1px solid black; padding: 2px;");

this->display->setAlignment(Qt::AlignRight);

this->display->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

this->layout->addWidget(this->display, 0, 0, 1, 3);

auto btn = new QPushButton(QString("="), this->\_centralWidget);

btn->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

QObject::connect(btn, &QPushButton::clicked, std::bind(&MainWindow::on\_click, this, QString("=")));

this->buttons->push\_front(btn);

this->layout->addWidget(btn, 0, 3);

int row\_index = 1;

for (auto row: btns\_map){

int col\_index = 0;

for (auto cell: row){

auto btn = new QPushButton(QString(cell), this->\_centralWidget);

btn->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

QObject::connect(btn, &QPushButton::clicked, std::bind(&MainWindow::on\_click, this, QString(cell)));

this->buttons->push\_front(btn);

this->layout->addWidget(btn, row\_index, col\_index);

col\_index++;

}

row\_index++;

}

}

void MainWindow::on\_click(QString value) {

qInfo() << value;

if (value.indexOf("=") >= 0) return;

if (value.indexOf("C") >= 0) {

this->display->setText("0");

return;

}

if (this->display->text() == QString("0"))

this->display->setText(value);

else

this->display->setText(this->display->text() + value);

}

MainWindow::~MainWindow() {

for (auto button: \*this->buttons)

delete button;

delete this->buttons;

delete this->display;

delete this->layout;

delete this->\_centralWidget;

}

**Paint**

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QApplication>

#include <QMainWindow>

#include <QMenuBar>

#include <QMenu>

#include <QAction>

#include <QToolBar>

#include <QPixmap>

#include <QKeySequence>

#include <QLabel>

#include <QScrollArea>

#include <QColor>

#include <QFileDialog>

#include <QMessageBox>

#include <QPainter>

#include <QMouseEvent>

#include <QPoint>

#include <QDebug>

#include <QColorDialog>

#include <QToolBar>

#include <QTextStream>

#include <QClipboard>

#include "canvas.h"

class MainWindow : public QMainWindow

{

Q\_OBJECT

enum STATES {

MAIN\_COLOR = 0,

BACKGROUND\_COLOR = 1,

COLOR\_CHOOSING = 2,

LINE\_WIDTH\_CHOOSING = 4,

PENCIL = 8,

LINE = 16,

RECTANGLE = 32,

CIRCLE = 64,

DRAWING = 128,

CLEAR\_INSTRUMENT = 0x87 // 100001111 - 0x87

};

QString filename;

QString filename\_save = nullptr;

bool isSaveNeeded = false;

QPixmap \*mainBuffer = nullptr, \*tempBuffer = nullptr, \*displayBuffer = nullptr;

Canvas \*displayLabel;

QScrollArea \*displayArea;

size\_t width;

size\_t height;

size\_t lineWidth;

QColor mainColor, backgroundColor;

int baseX, baseY;

int state;

QMenuBar \*menuBar;

QMenu \*fileMenu;

QAction \*newAction;

QAction \*openAction;

QAction \*saveAction;

QAction \*saveAsAction;

QAction \*copyToClipboard;

QAction \*pasteFromClipboard;

QAction \*exitAction;

QMenu \*toolsMenu;

QAction \*useMainColorAction;

QAction \*useBackgroundColorAction;

QAction \*chooseColorAction;

QAction \*chooseLineWidthAction;

QAction \*usePencil;

QAction \*useLine;

QAction \*useRectangle;

QAction \*useCircle;

QToolBar \*toolBar;

QClipboard \*clipboard;

void clear(int width, int height);

const QPixmap &makeColorIcon(const QColor& color);

void draw\_buffer(QPixmap \*pixmap);

private slots:

void on\_new();

void on\_open();

void on\_save();

void on\_save\_as();

void on\_copy\_to\_clipboard();

void on\_paste\_from\_clipboard();

void on\_swap\_color();

void on\_choose\_color();

void on\_choose\_line\_width();

void on\_use\_pencil();

void on\_use\_line();

void on\_use\_rectangle();

void on\_use\_circle();

void on\_mouse\_pressed(QPoint pos);

void on\_mouse\_released(QPoint pos);

void on\_mouse\_dragged(QPoint pos);

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

};

#endif // MAINWINDOW\_H

**Mainwindow.cpp**

#include "mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

this->setWindowTitle(QString("Paint"));

this->fileMenu = new QMenu("&Файл", this);

this->newAction = this->fileMenu->addAction("&Новый", QKeySequence("Ctrl+N"), this, &MainWindow::on\_new);

this->openAction = this->fileMenu->addAction("&Открыть", QKeySequence("Ctrl+O"), this, &MainWindow::on\_open);

this->saveAction = this->fileMenu->addAction("&Сохранить", QKeySequence("Ctrl+S"), this, &MainWindow::on\_save);

this->saveAsAction = this->fileMenu->addAction("Сохранить &как", QKeySequence("Ctrl+Shift+S"), this, &MainWindow::on\_save\_as);

this->fileMenu->addSeparator();

this->copyToClipboard = this->fileMenu->addAction("Ско&пировать в буфер обмена", QKeySequence("Ctrl+C"), this, &MainWindow::on\_copy\_to\_clipboard);

this->pasteFromClipboard = this->fileMenu->addAction("&Вставить из буфера обмена", QKeySequence("Ctrl+V"), this, &MainWindow::on\_paste\_from\_clipboard);

this->fileMenu->addSeparator();

this->exitAction = this->fileMenu->addAction("&Выйти", QKeySequence("Ctrl+Q"), QApplication::instance(), &QApplication::exit);

this->toolsMenu = new QMenu("&Инструменты", this);

this->useMainColorAction = this->toolsMenu->addAction("Основной &цвет", QKeySequence("Ctrl+W"), this, &MainWindow::on\_swap\_color);

this->useBackgroundColorAction = this->toolsMenu->addAction("Цвет &фона", QKeySequence("Ctrl+E"), this, &MainWindow::on\_swap\_color);

this->toolsMenu->addSeparator();

this->chooseColorAction = this->toolsMenu->addAction("Изменить ц&вет", QKeySequence("Ctrl+A"), this, &MainWindow::on\_choose\_color);

this->chooseLineWidthAction = this->toolsMenu->addAction("Изменить &ширину штриховки", QKeySequence("Ctrl+D"), this, &MainWindow::on\_choose\_line\_width);

this->toolsMenu->addSeparator();

this->usePencil = this->toolsMenu->addAction(QIcon(":/icons/pencil.png"), "&Карандаш", QKeySequence("Ctrl+P"), this, &MainWindow::on\_use\_pencil);

this->useLine = this->toolsMenu->addAction(QIcon(":/icons/line.png"), "&Линия", QKeySequence("Ctrl+L"), this, &MainWindow::on\_use\_line);

this->useRectangle = this->toolsMenu->addAction(QIcon(":/icons/rectangle.png"), "&Прямоугольник", QKeySequence("Ctrl+R"), this, &MainWindow::on\_use\_rectangle);

this->useCircle = this->toolsMenu->addAction(QIcon(":/icons/circle.png"), "&Окружность", QKeySequence("Ctrl+C"), this, &MainWindow::on\_use\_circle);

this->menuBar = new QMenuBar(this);

this->menuBar->addMenu(this->fileMenu);

this->menuBar->addMenu(this->toolsMenu);

this->setMenuBar(this->menuBar);

this->setMenuBar(this->menuBar);

this->displayArea = new QScrollArea;

this->displayArea->setWidgetResizable(true);

this->displayLabel = new Canvas(this);

this->displayLabel->setAlignment(Qt::AlignTop);

this->displayArea->setWidget(this->displayLabel);

connect(this->displayLabel, SIGNAL(mouse\_pressed(QPoint)), this, SLOT(on\_mouse\_pressed(QPoint)));

connect(this->displayLabel, SIGNAL(mouse\_released(QPoint)), this, SLOT(on\_mouse\_released(QPoint)));

connect(this->displayLabel, SIGNAL(mouse\_dragged(QPoint)), this, SLOT(on\_mouse\_dragged(QPoint)));

this->setCentralWidget(this->displayArea);

this->toolBar = new QToolBar("Инструменты", this);

this->addToolBar(this->toolBar);

this->toolBar->addAction(this->useMainColorAction);

this->toolBar->addAction(this->useBackgroundColorAction);

this->toolBar->addSeparator();

this->toolBar->addAction(this->chooseColorAction);

this->toolBar->addAction(this->chooseLineWidthAction);

this->toolBar->addSeparator();

this->toolBar->addAction(this->usePencil);

this->toolBar->addAction(this->useLine);

this->toolBar->addAction(this->useRectangle);

this->toolBar->addAction(this->useCircle);

this->clear(640, 480);

}

MainWindow::~MainWindow() {

delete this->mainBuffer;

delete this->tempBuffer;

delete this->menuBar;

delete this->fileMenu;

delete this->newAction;

delete this->openAction;

delete this->saveAction;

delete this->saveAsAction;

delete this->exitAction;

delete this->toolsMenu;

delete this->useMainColorAction;

delete this->useBackgroundColorAction;

delete this->chooseColorAction;

delete this->chooseLineWidthAction;

delete this->usePencil;

delete this->useLine;

delete this->useRectangle;

delete this->useCircle;

}

void MainWindow::clear(int width, int height)

{

this->width = width;

this->height = height;

this->mainColor = QColor(0, 0, 0, 255);

this->useMainColorAction->setIcon(this->makeColorIcon(this->mainColor));

this->backgroundColor = QColor(255, 255, 255, 255);

this->useBackgroundColorAction->setIcon(this->makeColorIcon(this->backgroundColor));

if (this->mainBuffer != nullptr) delete this->mainBuffer;

if (this->displayBuffer != nullptr) delete this->displayBuffer;

this->mainBuffer = new QPixmap(this->width, this->height);

this->mainBuffer->fill(this->backgroundColor);

this->displayBuffer = new QPixmap(this->width, this->height);

this->displayBuffer->fill(this->backgroundColor);

this->displayLabel->setPixmap(\*(this->displayBuffer));

}

const QPixmap &MainWindow::makeColorIcon(const QColor &color)

{

QPixmap \*icon = new QPixmap(16, 16);

icon->fill(color);

return \*icon;

}

void MainWindow::on\_new()

{

this->clear(640, 480);

this->isSaveNeeded = true;

}

void MainWindow::on\_open()

{

QString filename = QFileDialog::getOpenFileName(

this,

"Открыть изображение",

"",

""

);

if (filename.isEmpty()) return;

QPixmap \*image = new QPixmap(filename);

if (image->isNull()){

QMessageBox::critical(

this,

"Ошибка чтения",

"Не удалось открыть файл " + filename

);

return;

}

this->clear(image->width(), image->height());

QPainter p1(this->mainBuffer);

p1.drawPixmap(0, 0, this->width, this->height, \*image);

p1.end();

this->draw\_buffer(this->mainBuffer);

this->isSaveNeeded = true;

}

void MainWindow::on\_save()

{

if (!this->isSaveNeeded) return;

qInfo() << this->filename\_save;

if (this->filename\_save.isEmpty()) {

this->on\_save\_as();

return;

}

if (!this->mainBuffer->save(this->filename\_save)) {

QMessageBox::critical(

this,

"Ошибка сохранения",

"Не удалось сохранить файл в " + this->filename\_save

);

return;

}

this->isSaveNeeded = false;

}

void MainWindow::on\_save\_as()

{

QString filename = QFileDialog::getSaveFileName(

this,

"Сохранить файл как",

"",

""

);

if (filename.isEmpty()) return;

if (QFileInfo(filename).suffix().isEmpty()) {

filename += ".png";

}

this->filename\_save = filename;

qInfo() << this->filename\_save;

this->on\_save();

}

void MainWindow::on\_copy\_to\_clipboard()

{

this->clipboard = QGuiApplication::clipboard();

this->clipboard->setPixmap(\*this->mainBuffer);

}

void MainWindow::on\_paste\_from\_clipboard()

{

this->clipboard = QGuiApplication::clipboard();

QPixmap pm = this->clipboard->pixmap();

if (!pm.isNull()) {

this->tempBuffer = new QPixmap(this->width, this->height);

QPainter p(this->tempBuffer);

p.drawPixmap(0, 0, pm);

p.end();

this->state &= ~STATES::DRAWING;

QPixmap \*t = this->mainBuffer;

this->mainBuffer = this->tempBuffer;

delete t;

this->tempBuffer = nullptr;

this->draw\_buffer(this->mainBuffer);

this->isSaveNeeded = true;

} else {

QMessageBox::information(

this,

"Путота",

"В буфере нет изображения"

);

}

}

void MainWindow::on\_swap\_color()

{

QColor t = this->mainColor;

this->mainColor = this->backgroundColor;

this->backgroundColor = t;

this->useMainColorAction->setIcon(this->makeColorIcon(this->mainColor));

this->useBackgroundColorAction->setIcon(this->makeColorIcon(this->backgroundColor));

}

void MainWindow::on\_choose\_color()

{

QColor color = this->mainColor;

if (this->state & STATES::BACKGROUND\_COLOR)

color = this->backgroundColor;

color = QColorDialog::getColor(color, this, "Выберете цвет");

if (color.isValid()) {

if (this->state & STATES::BACKGROUND\_COLOR)

this->backgroundColor = color;

else

this->mainColor = color;

}

this->useMainColorAction->setIcon(this->makeColorIcon(this->mainColor));

this->useBackgroundColorAction->setIcon(this->makeColorIcon(this->backgroundColor));

}

void MainWindow::on\_choose\_line\_width()

{

}

void MainWindow::on\_use\_pencil()

{

this->state &= STATES::CLEAR\_INSTRUMENT;

this->state |= STATES::PENCIL;

}

void MainWindow::on\_use\_line()

{

this->state &= STATES::CLEAR\_INSTRUMENT;

this->state |= STATES::LINE;

}

void MainWindow::on\_use\_rectangle()

{

this->state &= STATES::CLEAR\_INSTRUMENT;

this->state |= STATES::RECTANGLE;

}

void MainWindow::on\_use\_circle()

{

this->state &= STATES::CLEAR\_INSTRUMENT;

this->state |= STATES::CIRCLE;

}

void MainWindow::on\_mouse\_pressed(QPoint pos)

{

this->state |= STATES::DRAWING;

this->baseX = pos.x();

this->baseY = pos.y();

if (this->tempBuffer != nullptr) delete this->tempBuffer;

this->tempBuffer = new QPixmap(this->width, this->height);

QPainter p(this->tempBuffer);

p.drawPixmap(0, 0, \*this->mainBuffer);

p.end();

}

void MainWindow::on\_mouse\_released(QPoint pos)

{

this->state &= ~STATES::DRAWING;

QPixmap \*t = this->mainBuffer;

this->mainBuffer = this->tempBuffer;

delete t;

this->tempBuffer = nullptr;

this->draw\_buffer(this->mainBuffer);

this->isSaveNeeded = true;

}

void MainWindow::on\_mouse\_dragged(QPoint pos)

{

QPainter p(this->tempBuffer); // если кидать буферы в стек, то можно реализовать Ctrl+Z

p.setPen(this->mainColor);

p.setBrush(this->backgroundColor);

if (this->state & STATES::PENCIL) {

p.drawLine(this->baseX, this->baseY, pos.x(), pos.y());

this->baseX = pos.x();

this->baseY = pos.y();

}

if (this->state & STATES::LINE) {

p.drawPixmap(0, 0, \*this->mainBuffer);

p.drawLine(this->baseX, this->baseY, pos.x(), pos.y());

}

if (this->state & STATES::RECTANGLE) {

p.drawPixmap(0, 0, \*this->mainBuffer);

QColor transparent\_color = this->backgroundColor;

transparent\_color.setAlpha(0);

p.setBrush(transparent\_color);

p.drawRect(this->baseX, this->baseY, pos.x() - this->baseX, pos.y() - this->baseY);

p.setBrush(this->backgroundColor);

}

if (this->state & STATES::CIRCLE) {

p.drawPixmap(0, 0, \*this->mainBuffer);

QColor transparent\_color = this->backgroundColor;

transparent\_color.setAlpha(0);

p.setBrush(transparent\_color);

p.drawEllipse(this->baseX, this->baseY, pos.x() - this->baseX, pos.y() - this->baseY);

p.setBrush(this->backgroundColor);

}

p.end();

this->draw\_buffer(this->tempBuffer);

}

void MainWindow::draw\_buffer(QPixmap \*pixmap)

{

QPainter p(this->displayBuffer);

p.drawPixmap(0, 0, \*pixmap);

p.end();

this->displayLabel->setPixmap(\*this->displayBuffer);

this->displayLabel->update();

}

**Animation ship**

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

#include <QPixmap>

#include <QPainter>

#include <QPaintEvent>

#include <QColor>

#include <QPointF>

#include <QBrush>

#include <QPen>

#include <QTransform>

#include <QPainterPath>

#include <QPolygonF>

#include <QLinearGradient>

#include <QRandomGenerator>

#include <QTime>

#include <cmath>

#include <vector>

#define LAMBDA 200.0

#define AMPLITUDE 20.0

#define LEVEL 0.7

#define SHIP\_WIDTH 200

#define SHIP\_HEIGHT 200

class MainWindow : public QMainWindow

{

Q\_OBJECT

QRandomGenerator\* rand;

QPixmap \*ship, \*sun, \*background;

void drawShip();

void drawSun();

double wawe\_phase, sun\_rotation\_phase, sun\_move\_phase;

QTimer \*wawe\_timer, \*sun\_rotation\_timer, \*sun\_move\_timer;

double wawe\_calc(double x);

QColor mix\_colors(QColor color1, QColor color2, double rate);

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

void paintEvent(QPaintEvent \*event) override;

public slots:

void onWaweTimer();

void onSunRotationTimer();

void onSunMoveTimer();

};

#endif // MAINWINDOW\_H

**Mainwindow.cpp**

#include "mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

this->resize(640, 480);

this->setWindowTitle("Animation");

this->wawe\_timer = new QTimer();

this->sun\_rotation\_timer = new QTimer();

this->sun\_move\_timer = new QTimer();

connect(this->wawe\_timer, &QTimer::timeout, this, &MainWindow::onWaweTimer);

connect(this->sun\_rotation\_timer, &QTimer::timeout, this, &MainWindow::onSunRotationTimer);

connect(this->sun\_move\_timer, &QTimer::timeout, this, &MainWindow::onSunMoveTimer);

this->wawe\_phase = 0;

this->sun\_rotation\_phase = 0;

this->sun\_move\_phase = 0;

this->rand = new QRandomGenerator(QTime::currentTime().msec());

this->drawSun();

this->drawShip();

this->wawe\_timer->start(50);

this->sun\_rotation\_timer->start(50);

this->sun\_move\_timer->start(50);

}

MainWindow::~MainWindow() {

delete this->ship;

this->sun\_rotation\_timer->stop();

delete this->sun\_rotation\_timer;

this->sun\_move\_timer->stop();

delete this->sun\_move\_timer;

delete this->sun;

this->wawe\_timer->stop();

delete this->wawe\_timer;

delete this->rand;

}

void MainWindow::drawShip() {

this->ship = new QPixmap(SHIP\_WIDTH, SHIP\_HEIGHT);

this->ship->fill(Qt::transparent);

QPainter p;

p.begin(this->ship);

QPainterPath pipes;

QPolygonF pipe1\_polygon;

pipe1\_polygon << QPointF(42, 145)

<< QPointF(42, 92)

<< QPointF(67, 92)

<< QPointF(67, 145);

pipes.addPolygon(pipe1\_polygon);

QPolygonF pipe2\_polygon;

pipe2\_polygon << QPointF(78, 145)

<< QPointF(78, 78)

<< QPointF(109, 78)

<< QPointF(109, 145);

pipes.addPolygon(pipe2\_polygon);

QPolygonF pipe3\_polygon;

pipe3\_polygon << QPointF(121, 145)

<< QPointF(121, 99)

<< QPointF(145, 99)

<< QPointF(145, 145);

pipes.addPolygon(pipe3\_polygon);

QLinearGradient pipes\_gradient(42, 92, 42, 145); // Направление сверху вниз

pipes\_gradient.setColorAt(0.0, QColor(90, 70, 70)); // Тёмный верх

pipes\_gradient.setColorAt(0.5, QColor(70, 50, 50)); // Очень тёмный центр

pipes\_gradient.setColorAt(1.0, QColor(60, 40, 40)); // Почти чёрный низ

p.setBrush(pipes\_gradient);

p.setPen(QPen(QColor(60, 60, 60), 1)); // Тёмная обводка

p.drawPath(pipes);

QPainterPath build;

QPolygonF build\_polygon;

build\_polygon << QPointF(60, 145)

<< QPointF(60, 115)

<< QPointF(121, 115)

<< QPointF(121, 145);

build.addPolygon(build\_polygon);

QLinearGradient build\_gradient;

build\_gradient.setColorAt(0.0, QColor(160, 160, 160));

build\_gradient.setColorAt(0.5, QColor(140, 140, 140));

build\_gradient.setColorAt(1.0, QColor(130, 130, 130));

p.setBrush(build\_gradient);

p.setPen(Qt::NoPen);

p.drawPath(build);

p.setPen(QColor(110, 110, 110));

const int window\_margin = 10;

bool fl\_y = false;

bool fl\_b = false;

for(int i = 0; i < 5; ++i){

int r = this->rand->bounded(0, 100);

if (r % 3){

p.setBrush(QColor(100, 149, 237));

fl\_b = true;

} else {

p.setBrush(QColor(255, 220, 51));

fl\_y = true;

}

if (i == 4 && !fl\_y) p.setBrush(QColor(255, 220, 51));

if (i == 4 && !fl\_b) p.setBrush(QColor(100, 149, 237));

p.drawEllipse(65 + i \* window\_margin, 120, 7, 7);

}

QPainterPath body;

QPolygonF body\_polygon;

body\_polygon << QPointF(15, 115)

<< QPointF(35, 115)

<< QPointF(55, 135)

<< QPointF(125, 135)

<< QPointF(152, 118)

<< QPointF(188, 116)

<< QPointF(165, 185)

<< QPointF(33, 185)

<< QPointF(18, 135)

<< QPointF(15, 115);

body.addPolygon(body\_polygon);

QLinearGradient body\_gradient(0, 125, 0, 180);

body\_gradient.setColorAt(0, QColor(80, 80, 80)); // Тёмно-серый

body\_gradient.setColorAt(1, QColor(30, 30, 30)); // Почти чёрный

p.setBrush(body\_gradient);

p.setPen(QPen(Qt::black, 2));

p.drawPath(body);

p.end();

}

void MainWindow::drawSun()

{

this->sun = new QPixmap(140, 140);

this->sun->fill(Qt::transparent);

QPainter p;

p.begin(this->sun);

p.setPen(QColor(244, 169, 0));

int sunX = 35;

int sunY = 35;

int sun\_width = 70;

int sun\_height = 70;

p.setBrush(QColor(253, 233, 16));

p.drawEllipse(sunX, sunY, sun\_width, sun\_height);

int ray\_count = 16;

int ray\_length = 25;

int ray\_width = 6;

int centerX = (2 \* sunX + sun\_width) / 2;

int centerY = (2 \* sunY + sun\_height) / 2;

for(int i = 0; i < ray\_count; ++i){

double place\_angle = 2 \* M\_PI / ray\_count;

QTransform t;

t.translate(centerX, centerY);

t.rotateRadians(i \* place\_angle);

t.translate(sun\_width / 2 + 5, -ray\_width / 2);

p.setTransform(t);

p.drawRect(0, 0, ray\_length, ray\_width);

}

p.end();

}

double MainWindow::wawe\_calc(double x)

{

return LEVEL \* this->height() + AMPLITUDE \* std::sin(x / LAMBDA \* M\_PI + wawe\_phase);

}

// Выкидывает проможуточные цвета (по умному - интерполяция цвета), здесь 0 - первый цвет, 1 - второй цвет, всё зависит от rate

QColor MainWindow::mix\_colors(QColor color1, QColor color2, double rate)

{

double r = (color2.redF() - color1.redF()) \* rate + color1.redF();

double g = (color2.greenF() - color1.greenF()) \* rate + color1.greenF();

double b = (color2.blueF() - color1.blueF()) \* rate + color1.blueF();

return QColor::fromRgbF(r, g, b);

}

void MainWindow::paintEvent(QPaintEvent \*event) {

QPainter p;

p.begin(this);

p.setRenderHint(QPainter::Antialiasing); // Сглаживане

// Sun move

double sunX = this->width() - this->width() \* std::sin(this->sun\_move\_phase / 2.0);

double sunY = (LEVEL \* this->height() - this->sun->height() / 2.0) - std::sin(this->sun\_move\_phase) \* ((LEVEL \* this->height() - this->sun->height() / 2.0));

// It's фон

// 0 — верх, 1 — низ

double t\_h = sunY / double(this->height());

QColor dawnT = QColor("#FFCF87");

QColor dawnB = QColor("#FFEFCF");

QColor noonT = QColor("#87CEEB");

QColor noonB = QColor("#E0FFFF");

QColor duskT = QColor("#FF4500");

QColor duskB = QColor("#8B0000");

// Рассвет - день - закат

QColor top, bottom;

if (t\_h < 0.5) {

// От рассвета к зенитц

double f = t\_h / 0.5;

top = mix\_colors(noonT, dawnT, f);

bottom = mix\_colors(noonB, dawnB, f);

} else {

// От зениту к закату

double f = (t\_h - 0.5) / 0.5;

top = mix\_colors(duskT, noonT, f);

bottom = mix\_colors(duskB, noonB, f);

}

// Градиент "от солнца"

QPointF centerr(width()/2.0, height()/2.0);

QPointF dir = QPointF(sunX, sunY) - centerr; // Типо вектор к солнышку

QPointF start = centerr + dir \* 2.0; // умножаем для красоты (наугад) :)

QPointF end = centerr - dir \* 2.5;

QLinearGradient background\_grad(start, end);

background\_grad.setColorAt(0.0, top);

background\_grad.setColorAt(1.0, bottom);

p.setPen(Qt::NoPen);

p.setBrush(background\_grad);

p.drawRect(rect());

// Sun rotation

QPointF center(sunX + this->sun->width()/2.0, sunY + this->sun->height()/2.0);

p.save();

p.translate(center);

double angle\_deg = this->sun\_rotation\_phase \* 180.0 / M\_PI; // Делаем угол из радиан в градусы

p.rotate(angle\_deg);

p.translate(-this->sun->width()/2.0, -this->sun->height()/2.0);

p.drawPixmap(0, 0, \*this->sun);

p.restore();

// Кораблик делает вжух-вжух лево-право

double x1 = (this->width() - SHIP\_WIDTH) / 2;

double x2 = x1 + SHIP\_WIDTH;

double y1 = this->wawe\_calc(x1);

double y2 = this->wawe\_calc(x2);

double alpha = std::atan((y2 - y1) / (x2 - x1));

QTransform ship\_t;

ship\_t.rotateRadians(alpha);

QPixmap \_ship = this->ship->transformed(ship\_t);

p.drawPixmap(

(this->width() - \_ship.width()) / 2,

this->height() \* LEVEL - \_ship.height() + 2 \* AMPLITUDE,

\_ship

);

// Волны

p.setPen(QColor("#00ffff"));

p.setBrush(QColor("#00ffff"));

std::vector<QPointF> points;

for (int x = 0; x < this->width(); ++x) {

double y = this->wawe\_calc(x);

points.push\_back(QPointF(x, y));

}

points.push\_back(QPointF(this->width(), this->height()));

points.push\_back(QPointF(0, this->height()));

p.drawPolygon(points.data(), points.size());

p.end();

}

void MainWindow::onWaweTimer() {

wawe\_phase += 0.1;

if (std::fabs(this->wawe\_phase - 2 \* M\_PI) < 0.1)

this->wawe\_phase = 0;

this->update();

}

void MainWindow::onSunRotationTimer()

{

this->sun\_rotation\_phase += 0.05;

if (std::fabs(this->sun\_rotation\_phase - 2 \* M\_PI) < 0.05)

this->sun\_rotation\_phase = 0;

this->update();

}

void MainWindow::onSunMoveTimer()

{

this->sun\_move\_phase += 0.01;

if (std::fabs(this->sun\_move\_phase - 1 \* M\_PI) < 0.01)

this->sun\_move\_phase = 0;

this->update();

}