Дз 8

1. *Типы выражений в с++*

- **Первичные выражения** — исходные компоненты, из которых состоят все остальные выражения (например, оператор область разрешения (`::`).
- **Выражения постфикса** — состоят из основных выражений или выражений, в которых постфиксные операторы (например, вызова функции (), доступа к членам - > и тд) следуют за основным выражением.
- **Выражения, сформированные с унарными операторами**. Унарные операторы действуют только на один операнд в выражении (например, унарный плюс, оператор логического отрицания).
- **Выражения, сформированные двоичными операторами** — операторы действуют на два операнда в выражении (например, выражения с операторами умножения, сложения и тд).
- **Выражения с условным оператором ?: —** принимает три операнда.
- **Константные выражения** — целиком состоят из константных данных.
- **Выражения с явными преобразованиями типов.**

```
// Пример
int x = 7;
float y;
y = float( x );
```

- **Сведения о типе времени выполнения (RTTI)** — это механизм, позволяющий определить тип объекта во время выполнения программы.

Или вы имели в виду rvalue/lvalue-выражения?

2. *Зачем нужны rvalue ссылки. Какие аргументы можно передавать по каким типам ссылок.*

rvalue ссылки — это ссылки на временные объекты/выражения, которые не имеют идентификатора в памяти, они нужны для оптимизации кода, так как позволяют вместо копирования осуществлять процедуру перемещения.

Аргументы можно передавать по следующим типам ссылок:

- lvalue ссылки: используются для передачи объектов, у которых есть идентификатор в памяти, и могут быть изменены внутри функции.
- rvalue ссылки: используются для передачи временных объектов/выражений.

3. *Семантика перемещения и копирования. Какими средствами языка реализуются.*

Семантика перемещения — способ передачи ресурсов из одного объекта в другой без реального копирования данных.

Семантика копирования — способ передачи ресурсов из одного объекта в другой путем создания копии данных.

Семантика перемещения реализуется с помощью `move constructor` и `move assignment`, а семантика копирования реализуется с помощью `copy constructor` и `copy assignment`. Кроме того, в C++ можно определять пользовательские версии этих операторов для своих собственных типов данных.

4. *Что делает std::move.*

`std::move()` выполняет преобразование lvalue в rvalue, что позволяет перемещать ресурсы из одного объекта в другой вместо их копирования.

5. *Правила 3х и 5ти, когда необходима пользовательская реализация специальных функций членов.*

Правило 3х: если класс имеет пользовательскую реализацию конструктора копирования, оператора присваивания или деструктора, то почти всегда необходимо также реализовать оставшиеся функции, чтобы избежать проблем с утечками памяти или неправильным поведением объектов при копировании и присваивании.

Правило 5ти — это расширенное правило 3х: если класс имеет пользовательскую реализацию конструктора копирования, оператора присваивания, деструктора, конструктора перемещения или оператора перемещения, то почти всегда необходимо также реализовать оставшиеся функции, чтобы избежать проблем с утечками памяти или неправильным поведением объектов при копировании и присваивании.

6. *Сигнатура копирующих и перемещающих специальных функций членов и операторов.*

```
class ClassName {
public:
ClassName(const ClassName&); // копирующий конструктор
ClassName(ClassName&&); // перемещающий конструктор
ClassName&operator=(const ClassName&); // копирующий оператор присваивания
ClassName&operator=(ClassName&&); // перемещающий оператор присваивания
};
// ClassName - имя класса, const ClassName& - ссылка на константный объект класса, ClassNa
me&& - rvalue ссылка на объект класса.
```

7. *Как спровоцировать выполнение перемещающих операций, для объектов предусматривающих такую возможность.*

Для того чтобы спровоцировать выполнение перемещающих операций для объектов класса, предусматривающих такую возможность, можно использовать стандартные функции и операции, такие как std::move() и оператор перемещения. Например, можно создать временный объект класса и передать его в качестве аргумента для перемещающего конструктора или оператора перемещения:

```
ClassName obj1; // создание объекта
ClassName obj2(std::move(obj1)); // вызов перемещающего конструктора
ClassName obj3; // создание еще одного объекта
obj3 = std::move(obj2); // вызов оператора перемещения
```

8. *Что произойдет при попытке вызова функции/метода принимающего не rvalue ссылку с аргументом rvalue ссылкой?*

При попытке вызова функции или метода, принимающего не rvalue ссылку с аргументом rvalue ссылкой в C++, будет произведено копирование объекта, а не перемещение, так как rvalue ссылка не может быть привязана к lvalue.

9. *Ключевые слова default и delite в объявлении специальных функций членов.*

Ключевые слова default и delete используются в объявлении специальных функций членов.

default используется для явного указания компилятору генерировать реализацию по умолчанию для конструкторов, деструкторов, операторов присваивания и операторов копирования.

delete используется для явного запрещения компилятору генерировать реализацию для конструкторов, деструкторов, операторов присваивания и операторов копирования.

10. *Return Value Optimization - Что такое.*

Return Value Optimization (RVO) — это оптимизация, которая позволяет компилятору оптимизировать возврат значения из функции, избегая лишних копирований объекта. Вместо создания временной копии объекта и возвращения ее, компилятор напрямую конструирует объект в месте, где он должен быть использован.

Эта оптимизация позволяет уменьшить накладные расходы на копирование объектов и улучшить производительность программы.

11. *Что значит "обработать ошибку". Какие существуют способы обработки ошибок, кроме исключений?*

Обработка ошибок в программировании — принятие мер для управления возникшей ошибкой или исключительной ситуацией.
Это может включать в себя вывод сообщения об ошибке, запись в лог, повторное

выполнение операции или принятие альтернативных действий.
Кроме использования исключений, существуют другие способы обработки ошибок:

- Возвращение специальных значений: функция может возвращать специальное значение, которое указывает на ошибку, например, -1 или null.
- Установка глобальной переменной ошибки: функция устанавливает значение глобальной переменной, которая затем проверяется вызывающим кодом.
- Использование флагов ошибок: функция устанавливает флаг ошибки, который затем проверяется вызывающим кодом.
- Обработка ошибок с помощью обратных вызовов: функция принимает указатель на функцию обратного вызова, которая будет вызвана в случае ошибки.
- Прекращение выполнения программы: в некоторых случаях возникновение критической ошибки может привести к завершению работы программы.

12. *Опишите механизм работы исключений. Достоинства и недостатки.*

Механизм работы исключений в программировании заключается в том, что при возникновении ошибки или исключительной ситуации в коде, программа генерирует исключение, которое затем может быть обработано в специальном блоке кода. Механизм работы исключений предполагает использование ключевых слов try, catch и throw для обработки ошибок. Код, который может вызвать ошибку, помещается в блок try, а код для обработки ошибки - в блок catch. Если возникает ошибка, то в блоке try генерируется исключение (throw), которое затем перехватывается блоком catch.

Достоинства использования исключений в C++:

- Упрощение структуры программы: исключения позволяют избавиться от множества проверок на ошибки в коде, что делает его более компактным и понятным.
- Возможность передачи информации об ошибке: исключения могут содержать информацию о типе и месте возникновения ошибки, что упрощает её обработку.

Недостатки использования исключений в C++:

- Производительность: механизм исключений может привести к потере производительности, поскольку требуется дополнительная работа для генерации и обработки исключений.
- Сложность в использовании: неправильное использование исключений может привести к неожиданным результатам и усложнить отладку программы.
- Необходимость поддержки языка: не все среды разработки и компиляторы поддерживают механизм исключений, что может создавать проблемы при переносе кода на другие платформы.

13. *Сворачивание стека, при выбрасывании исключений. Уничтожение переменных, оставшихся на стеке. Спецификатор noexcept, std::terminate().*

При выбрасывании исключений происходит сворачивание стека. Это означает, что все локальные переменные и объекты, созданные в блоке try, уничтожаются. Это делается для того, чтобы освободить память, занятую этими переменными, и предотвратить утечку памяти.

Для указания того, что функция не выбрасывает исключений, можно использовать спецификатор noexcept. Это позволяет компилятору оптимизировать код, так как он знает, что внутри функции не будет выброшено исключение.

Функция std::terminate() вызывается при необработанном исключении, то есть когда исключение не было перехвачено ни одним из блоков try-catch.
При вызове std::terminate() программа завершается аварийно.

14. *Как обрабатываются исключения различных типов(типы данных), множественность блоков catch.*

Исключения различных типов данных обрабатываются с помощью блоков catch, которые определены для каждого типа исключения. Например, если возникает исключение типа int, его можно перехватить с помощью блока catch с параметром int. Аналогично, для исключений других типов данных используются соответствующие блоки catch.

Множественность блоков catch позволяет обрабатывать различные типы исключений в одной конструкции try-catch. Это позволяет более гибко управлять

обработкой исключений в программе.

Кроме того, можно использовать стандартную библиотеку <exception> для создания пользовательских типов исключений и их обработки с помощью соответствующих блоков catch.

15. *Недостатки механизма исключений.*

- Накладные расходы: использование исключений может привести к дополнительным накладным расходам, поскольку компилятор должен генерировать код для обработки исключений, что может увеличить размер исполняемого файла и замедлить выполнение программы.
- Скрытые ошибки: иногда исключения могут быть перехвачены и обработаны без вывода сообщения об ошибке или логирования. Это может затруднить отладку программы и поиск причины возникновения ошибок.
- Необходимость корректной обработки: использование исключений требует от разработчиков корректной обработки исключений, в противном случае программа может завершиться аварийно или привести к утечке ресурсов.