Выдержка из Положения о проведения текущего контроля успеваемости и промежуточной аттестации обучающихся:

5.14. Экзамены и зачеты по усмотрению кафедры могут проводиться как в устной, так и в письменной форме по билетам или тестовым заданиям, утвержденным в установленном порядке. Экзаменатору предоставляется право задавать обучающимся дополнительные вопросы сверх вопросов билета, а также, помимо теоретических вопросов, давать для решения задачи и примеры, связанные с дисциплиной.

Вопросы для подготовки к экзамену

по дисциплине **«Программирование на языке высокого уровня»**

2 семестр

(вопросы из ФОС)

1. Структурные типы данных и переменные этих типов.
2. Передача структурных переменных функциям.
3. Указатели на структурные переменные.
4. Массивы структурных переменных.
5. Объединения.
6. Поля битов. Средство typedef.
7. Аргументы, используемые по умолчанию.
8. Использование ссылок.
9. Встроенные функции.
10. Операция разрешения видимости.
11. Перегруженные функции.
12. Особенности описания переменных в программах на C++.
13. Определение классов.
14. Элементы класса.
15. Правила обращения к элементам класса.
16. Конструктор и деструктор.
17. Конструктор копии. Операция присваивания.
18. Преобразование объектов класса в другие типы и их получение из других типов.
19. Получение представителей класса из переменных других типов. Конструктор преобразования.
20. Перегрузка операций.
21. Обработка исключительных ситуаций.

Список вопросов из ФОС (курсивом) с уточняющими вопросами по материалам лекций

1. *Структурные типы данных и переменные этих типов.*
2. *Передача структурных переменных функциям.*
3. *Указатели на структурные переменные.*
4. *Массивы структурных переменных.*

1. Что такое структура и для чего она используется? Какие типы данных могут входить в состав структуры? Как объявляется структура и переменная структурного типа?
2. Как осуществляется доступ к полям структуры?
3. В чем отличие обращения к полям структуры через переменную и через указатель?
4. Что представляет собой массив структур? Чем массив структур принципиально отличается от нескольких отдельных переменных?
5. Как осуществляется доступ к полям элементов массива структур?
6. Почему массив структур удобно использовать для описания списков, таблиц и записей?
7. Как осуществляется копирование структур при присваивании? Какие ограничения и особенности имеет копирование структур в C++?

5. *Объединения.*
6. *Поля битов. Средство typedef.*

1. Что такое объединение (union)? Чем объединение отличается от структуры (struct)? Как определяется размер объединения?
2. В каких случаях целесообразно использовать объединения?
3. Что такое битовое поле? Как объявляется битовое поле в структуре?
4. Зачем в структуре может быть предусмотрено поле reserved?
5. Что такое padding и зачем он добавляется компилятором? Почему после поля int в структуре может появляться промежуточный padding?
6. Поясните итоговый размер структуры по примеру, предложенному преподавателем.
7. Поясните роль операторов typedef/using. В чём отличие typedef от using? Как использование псевдонимов повышает читаемость кода?

7. *Аргументы, используемые по умолчанию.*
8. *Использование ссылок.*
9. *Встроенные функции.*

1. Поясните назначение аргументов по умолчанию в C++? Как они объявляются? Каковы основные правила задания аргументов по умолчанию (порядок объявления, возможность пропуска)?
2. Почему нельзя задать значение по умолчанию только для первого параметра, оставив второй без значения? Приведите пример неправильного объявления.
3. Как компилятор подставляет значения по умолчанию во время вызова функции?
4. В чем разница между аргументами по умолчанию и перегрузкой функций? Приведите пример, когда их совместное использование приводит к неоднозначности.
5. Что такое ссылка в C++? Как она объявляется и чем отличается от указателя? Объясните, что произойдёт с исходной переменной, если изменить значение через ссылку.
6. В чём разница между передачей аргумента в функцию по значению и по ссылке? Приведите пример.
7. Для чего используются константные ссылки в параметрах функций? Какие преимущества они дают?

8. Каким образом функция может вернуть через ссылочный параметр дополнительную информацию (например, размер нового массива)?
9. С какой целью они применяются встроенные (inline) функции? Как объявить функцию встроенной? Приведите синтаксис.
10. Какие накладные расходы связаны с обычным вызовом функции? Перечислите основные шаги. В каких случаях использование inline-функций дает наибольший выигрыш в производительности?
11. Почему для длинных или сложных функций (с циклами, вводом-выводом) применение inline нецелесообразно?

10.Операция разрешения видимости.
11.Перегруженные функции.
12.Особенности описания переменных в программах на C++.

1. Что такое операция разрешения видимости :: и для каких целей она применяется? Как обратиться к глобальной переменной, если её имя скрыто локальной переменной с тем же именем?
2. Можно ли вызвать глобальную функцию, если её имя скрыто локальной переменной? Если да, то как?
3. Что называется перегруженной функцией в языке C++? По каким признакам могут различаться перегруженные функции? В чем состоит основная идея перегрузки функций?
4. Учитывается ли тип возвращаемого значения при различении перегруженных функций? Почему нельзя перегружать функции только по возвращаемому типу?
5. Как компилятор определяет, какую перегруженную функцию необходимо вызвать? Какие преимущества дает использование перегрузки функций?
6. Какие проблемы могут возникать при использовании параметров по умолчанию в перегруженных функциях?
7. Почему изменение структуры данных может привести к необходимости изменения функций?

13.Определение классов.
14.Элементы класса.
15.Правила обращения к элементам класса.
16.Конструктор и деструктор.

1. Что называется классом в языке C++? Какие элементы могут входить в состав класса? Каково назначение спецификатора доступа public? private?
2. Что называется интерфейсом класса? полями данных класса? методом класса? Как осуществляется вызов метода класса?
3. Что означает ключевое слово const, указанное после объявления метода? Почему методы, не изменяющие состояние объекта, рекомендуется объявлять с использованием const?
4. В чем состоит назначение директив #ifndef, #define и #endif в заголовочных файлах? Что называется макросом в языке C/C++?
5. В чем заключается принцип инкапсуляции? Какие преимущества дает использование инкапсуляции при разработке программ?

6. В чем различие между определением методов внутри класса и вне класса? Влияет ли размещение методов на область действия класса и правила доступа?
7. Какие преимущества имеет определение методов внутри класса? Какие недостатки имеет определение методов внутри класса? Какие преимущества имеет вынесение реализации методов за пределы класса? В каких случаях целесообразно использовать способ через ::?
8. Что такое конструктор класса? Каково назначение конструктора? Когда вызывается конструктор? Почему инициализация через конструктор предпочтительнее ручного присваивания?
9. Какова основная задача деструктора? Каков синтаксис деструктора? Когда вызывается деструктор? Почему деструктор не принимает параметров и не возвращает значение?
10. В каком порядке вызываются деструкторы объектов? В каких случаях деструктор является обязательным?
11. Что происходит при использовании динамической памяти без деструктора? Какие операции выполняются при работе с динамической памятью? Почему класс, использующий динамическую память, обязан освобождать ресурсы?
12. Как связаны между собой инкапсуляция, конструктор и деструктор?

17. Конструктор копии. Операция присваивания.

18. Преобразование объектов класса в другие типы и их получение из других типов.

1. Какую задачу решает конструктор копии? В каких ситуациях вызывается конструктор копии? Что такое глубокое копирование и чем оно отличается от поверхностного?
2. Почему недостаточно реализовать только конструктор копии?
3. В чем различие между созданием объекта, например, Student s2 = s1; и присваиванием s2 = s1;?
4. В какой последовательности должны выполняться действия при глубоком копировании?
5. В чем заключается назначение оператора присваивания? Почему оператор присваивания должен освобождать старую память? Зачем в операторе присваивания выполняется проверка на самоприсваивание?
6. Что понимается под конструктором преобразования в языке C++? Почему конструктор с одним параметром может выполнять неявное преобразование типов?
7. Какую задачу решает ключевое слово explicit? Как изменяется поведение программы при объявлении конструктора как explicit? Какие формы записи становятся недопустимыми при использовании explicit?
8. В чем различие между явным и неявным преобразованием типов?
9. Почему в практическом программировании рекомендуется использовать explicit для конструкторов преобразования?
10. Чем конструктор преобразования отличается от конструктора копии?
11. Что понимается под преобразованием типов для пользовательских классов в C++? Что такое оператор преобразования типа и какова его роль в классе? Каков синтаксис объявления оператора преобразования типа? В чем его особенности?

19. Получение представителей класса из переменных других типов. Конструктор преобразования.

20. Перегрузка операций.

21.Обработка исключительных ситуаций.

1. Что понимается под преобразованием типов для пользовательских классов в C++? Что такое оператор преобразования типа и какова его роль в классе? Каков синтаксис объявления оператора преобразования типа? В чем его особенности?
2. Почему у оператора преобразования типа отсутствует явно указанный возвращаемый тип?
3. В каких ситуациях происходит неявное преобразование объекта класса к другому типу?
4. Почему неявные преобразования считаются потенциально опасными?
5. В каких случаях целесообразно использовать операторы преобразования типов, а в каких — нет?
6. Что такое перегрузка операций в C++? В чем заключается отличие перегрузки операций от перегрузки функций? Какие операторы языка можно перегружать, а какие — нельзя?
7. Каким образом перегрузка операций связана с функциями языка C++? Каков общий синтаксис перегрузки бинарного оператора внутри класса?
8. Почему при перегрузке операторов необходимо сохранять их логический смысл?
9. Как происходит вызов перегруженного оператора в выражении a + b?
10. Почему перегрузка операторов повышает читаемость и выразительность кода?
11. В чем заключается ключевая идея механизма исключений в C++? Какова роль оператора throw? Что происходит после его выполнения?
12. Что такое исключение с точки зрения языка C++? Как устроена конструкция try–catch и для чего она используется?
13. Почему исключения особенно важны при работе с конструкторами? Что происходит, если в конструкторе возникает исключение?
14. Что понимается под ресурсом в языке C++? Приведите примеры.
15. Что такое идиома RAII и в чем ее основная идея? Как RAII обеспечивает корректное освобождение ресурсов при исключениях? Какие обязанности возлагаются на конструктор и деструктор в рамках RAII?
16. Для чего используется спецификатор explicit в конструкторе?
17. Почему необходим конструктор копирования при работе с динамической памятью?
18. В чем заключается проблема поверхностного копирования?
19. Как перегруженный оператор [] обеспечивает безопасный доступ к элементам массива?
20. С какой целью могут быть реализованы две версии оператора [] (const и неконстантная)?
21. В чем заключается интеграция классов, ресурсов и исключений в обеспечении устойчивости программы?