Санкт-Петербургский
государственный
университет

Вопросы и ответы к зачёту 1 апреля 2024 г.
№ 3

Санкт-Петербургский государственный университет

Вопросы

1. Какие виды тестов существуют и для чего они применяются?
2. Что дают модульные тесты разработчикам?
3. Какие критерии налагаются на модульные тесты?
4. Что такое TDD?
5. Расскажите о системах автоматизации сборки и тестирования проектов в C++.
6. Что такое статическая и динамическая библиотеки?
7. В чём плюсы и минусы статической и динамической библиотек?
8. В чем разница между исполняемым файлом и динамической библиотекой?
9. Что такое DLL hell?
10. Что должно быть реализовано в классе?

Санкт-Петербургский государственный университет

Вопросы

11. Что такое конструктор по умолчанию?
12. Для чего нужны default и delete с конструкторами и другими методами класса?
13. Что такое конструктор копирования и оператор присваивания копированием?
14. Что такое lvalue и rvalue?
15. Что такое конструктор перемещения и оператор присваивания перемещения?
16. В каких случаях надо опеределять КК, ОПК, КП, ОПП, деструктор?
17. В каких случаях КК, ОПК, КП, ОПП, деструктор будут созданы компилятором?
18. Что такое делегирующий конструктор?
19. Что такое наследование? Напишите синтаксис наследования класса.
20. Какие бывают типы наследования?

Санкт-Петербургский государственный университет

Вопросы

21. Какие особенности наследования конструкторов, операторы присваивания?
22. Какие особенности наследования деструктора?
23. Как запретить наследование класса?
24. Что такое полиморфизм? Какие механизмы связывания сущестуют в C++?
25. Что такое виртуальные функции и зачем они нужны?
26. Для чего используют виртуальный деструктор?
27. Что такое абстрактный класс и чистая виртуальная функция?
28. Может ли иметь реализацию pure virtual function?
29. Что такое множественное наследование? Какие у него проблемы?
30. Как решить проблему ромбовидного наследования?

Санкт-Петербургский государственный университет

Вопросы

31 — 42 Какой результат выполнения программы? Объясните.

Санкт-Петербургский государственный университет

Ответы

1 Какие виды тестов существуют и для чего они применяются?

Тесты бывают модульные (unit), функциональные и интеграционные.

Модульное (Unit) тестирование — это тестирование на корректность отдельных модулей (hpp + cpp) / классов исходного кода без взаимодействия / зависимости с другими модулями/классами.

Функциональное тестирование — это тестирование отдельных функциональностей в рамках одной независимой части системы. Пример: аутентификация пользователя, запрос в бд, отправка/обработка команды на сервер.

Интеграционное тестирование — это тестирование совместной работы нескольких частей системы (программных модулей, микросервисов, приложений распределённой системы). Пример: сценарий когда пользователь добавляет товар в корзину, затем оформляет его и оплачивает.

2 Что дают модульные тесты разработчикам?

1. Уменьшают время на общее тестирование (быстро находится большое количество мелких ошибок).
2. Защищают от регрессии.
3. Подтверждают документацию. Показывают как работает код.
4. Показывают связность кода: чем больше % покрытия кода тестами, тем менее связный код системы (и более легко заменяемый).

Санкт-Петербургский государственный университет

Ответы

3 Какие критерии налагаются на модульные тесты?

1. Быстрота

Модульных тестов тысячи. Они запускаются регулярно разработчиком для отсечения всех мелких ошибок.

2. Изолированность

Результат одного теста не должен влиять на результат другого теста. Модульные тесты должны запускаться параллельно.

3. Повтряемость

Не должно возникать ситуаций, когда 1 раз в 100 запусков тест будет падать. Трудно выявляемые и исправляемые ситуации.

4. Самопроверяемый

В тесте для запуска не надо ничего настраивать снаружи этого теста.

5. Уместный

Тест должен проверять то, что необходимо.

4 Что такое TDD?

TDD (Test Driving Development) — это техника разработки кода, при которой тесты пишутся до реализации:

1. Вначале создаётся структура класса в хедере и пустая реализация в соответствующей единице трансляции (.cpp).
2. Потом пишутся тесты, описывающие все необходимые ситуации с методом, который будет реализован.
3. Реализовывается соответствующий пункту 2 метод и запускаются тесты.
4. Пункты 2 и 3 повторяются пока не будут реализованы все методы класса.

Санкт-Петербургский государственный университет

Ответы

5 Расскажите о системах автоматизации сборки и тестирования проектов в C++.

Для автоматизирования сборки и тестирования проекта в C++ можно использовать связку Cmake + Ninja.

Cmake — кроссплатформенная автоматизированная система сборки проектов, созданная в 2000 г. для облегчения написания сценариев на скриптовом языке Make. Cmake является свободным и открытым ПО. Cmake не собирает исполняемые файлы, а лишь генерирует файлы сборки из предварительно написанного сценария CmakeLists.txt в проект одного из видов: make, ninja, visual studio и т. д. И только потом задействуется соответствующий собранному проекту инструмент для компиляции кода.

Для компиляции кода на следующем этапе после Cmake может быть использвана утилита ninja. Она является улучшенной и доработанной версией make.

Для запуска тестов cmake содержит встроенную утилиту ctest и специальный синтаксис Cmake скриптового языка.

6 Что такое статическая и динамическая библиотеки?

Статическая библиотека — это файл .lib для windows или .a для unix, который содержит набор объектных файлов функций и классов. Она используется в процессе статической линковки в исполняемый файл вместе с другим кодом и статическими библиотеками во время его создания.

Динамическая библиотека — это файл .dll (dynamic link library) для windows или .so для unix, который содержит набор объектных файлов функции и классов. Но линковка динамической библиотеки запускается в момент создания процесса (когда вы запускаете исполняемый файл на выполнение). Также возможна её линковка уже после запуска, т.е. библиотека может быть подгружена в адресное пространство ужа работающего процесса.

Санкт-Петербургский государственный университет

Ответы

7 В чём плюсы и минусы статической и динамической библиотек?

Плюсы статической библиотеки:

1. Простота установки пользователем

Приложение, использующее статическую библиотеку, может быть запущено на любом компьютере без необходимости установки дополнительных зависимостей.

2. Скорость работы

Т.к. функции и классы статической библиотеки встраиваются в исполняемый файл программы, то процесс выполнения (вызова этих функций и классов) будет более быстрым по сравнению с динамическим подходом.

3. Надёжность

Т.к. все необходимые функции и классы находятся в рамках одного файла, то версии библиотеки могут быть контролируемы и учитываться во время разработки.

Минусы статической библиотеки:

1. Большой размер исполняемого файла.

2. Требование перекомпиляции зависимых файлов при любом изменении в библиотеке.

Плюсы и минусы динамической библиотеки обратны минусам и плюсам статической библиотеки.

Дополнительным важным плюсом динамической библиотеки является то, что она загружается в память только тогда, когда надо. И также её можно выгрузить из процесса, когда надо.

Санкт-Петербургский государственный университет

Ответы

8 В чем разница между исполняемым файлом и динамической библиотекой?

1. Файлы exe — это исполняемые файлы, которые можно запускать независимо. А файлы dll — это программы динамически подключаемых библиотек, используемые для совместного использования кода и ресурсов.

2. Файлы exe содержат инструкции, которым компьютер следует, чтобы выполнить программу, а файлы dll содержат код (в виде машинного кода), который может использоваться несколькими программами одновременно.

9 Что такое DLL hell?

Все dll должны быть совместимы от версии к версии и взаимозаменяемы в обе стороны. На практике всё получилось наоборот. Это проблема несовместимости и невзаимозаменяемости в обе стороны разных версии dll и стала называться dll hell.

10 Что должно быть реализовано в классе?

1. Конструкторы.

1.1. Конструктор по умолчанию.

1.2. Конструкторы с параметрами.

1.3. Конструктор копирования.

1.4. Конструктор перемещения.

2. Деструктор.

3. Операторы присваивания.

3.1. Оператор присваивания копирования.

3.2. Оператор присваивания перемещения.

4. Другие методы

Санкт-Петербургский государственный университет

Ответы

11 Что такое конструктор по умолчанию?

КпоУ — это конструктор, который вызывается без передачи аргументов (включая конструктор с параметрами, имеющими значение по умолчанию):

LongNumber x;

LongNumber *xptr = new LongNumber();

Каждый класс может иметь только один конструктор по умолчанию либо без параметров, либо с параметрами, имеющими значения по умолчанию.

При объявлении массива объектов или объявлении динамически КпоУ инициализирует все элементы:

LongNumber arr[10];

LongNumber *arr = new LongNumber[10];

Если в классе не определён КпоУ, компилятор неявно создаст его. Он будет аналогичен явно объявленному конструктору с пустым телом.

Если есть определеные конструкторы, но среди них нет КпоУ, то компилятор его не создаст. И при вызове LongNumber x; будет ошибка компиляции.

12 Для чего нужны default и delete с конструкторами и другими методами класса?

LongNumber () = default; говорит компилятору явно создать КпоУ или другой метод (с реализацией по умолчанию). Использование конструкции с default бывает удобно, когда нам надо дать возможность создавать неинициализированные объекты, а среди полей класса есть поля с ключевым словом const или этих полей много (при этом реализован конструктор другого вида и компилятор в таком случае не будет создавать КпоУ).

LongNumber () = delete; удаляет КпоУ. Теперь любая ситуация, в которой вызывается КпоУ будет генерировать ошибку компиляции.
delete может быть использован по одношению к любому методу. Его используют, когда хотят **запретить** какие-то действия с объектами, например: создание неинициализированных объектов → LongNumber () = delete; запретить (присваивание) копирование объектов → LongNumber& operator = (const LongNumber& x) = delete; и т.п.

Санкт-Петербургский государственный университет

Ответы

13 Что такое конструктор копирования и оператор присваивания копированием?

КК и **ОПК** — это ситуации, когда надо создать копию существующего объекта или через инициализацию, или через присваивание. При этом, будет создано два разных объекта с одинаковыми данными, и эти данные будут храниться в разных областях памяти.

LongNumber(const LongNumber& x);

LongNumber& operator = (const LongNumber& x);

КК и ОПК создаются компилятором автоматически. Но если объект содержит указатель или ссылку, то их надо определять самостоятельно. Если этого не сделать, то будет несколько объектов, ссылающихся на одну и ту же область памяти. Что приведёт к неопределённому поведению программы.

Надо различать:

LongNumber x = y; — КК

и

LongNumber x;

x = y; — ОПК

Существует три **случая вызова КК**:

1. Когда объект является возвращаемым значением.

   LongNumber x = func();

2. Когда объект передается в функцию по значению в качестве аргумента.

   func(LongNumber x);

...

Санкт-Петербургский государственный университет

Ответы

3. Когда объект конструируется на основе другого объекта (того же класса).

```
LongNumber x = y;
```

Вопрос: Чем отличается конструктор копирования от оператора присваивания копирования?

КК применяется тогда, когда объект ещё не существует в памяти. А ОПК применяется к уже существующему в памяти объекту.

14 Что такое lvalue и rvalue?

В C++ используемые значения можно разделить на две группы: lvalue и rvalue.

Lvalue представляем ИМЕНОВАННОЕ значение (переменные, параметры, константы), с которыми ассоциирован АДРЕС ПАМЯТИ для хранения. Lvalue можно ПРИСВОИТЬ значение.

Rvalue — это НЕИМЕНОВАННОЕ значение т.е. то, что можно ТОЛЬКО ПРИСВАИВАТЬ, но ему нельзя присвоить (литералы, результаты выражений). Rvalue удаляется из памяти сразу после выполнения операции, в то время как Lvalue только в момент окончания своей области видимости.

```
int x = 5;
```

x — lvalue

5 — rvalue

rvalue-ссылка — это переменная, которая ссылается на результат выражения rvalue. Привязка продлевает время жизни такого rvalue до окончания области видимости.

```
int&& rvx {5};
```

rvx хранит ссылку на rvalue значение, но само является lvalue значением.

Санкт-Петербургский государственный университет

Ответы

15 Что такое конструктор перемещения и оператор присваивания перемещения?

КП позволяет переместить данные, принадлежащие объекту rvalue, в lvalue без копирования значений из динамической памяти (копируются значения указателей, но при этом надо не забыть присвоить копируемым указателям nullptr). Ещё КП применяется, когда копирование данных нежелательно или излишне.

```
LongNumber(LongNumber&& x);
```

ОПП решает ту же задачу, что и КП, но в ситуации, когда объект lvalue уже существует в памяти и ему присваивается новое значение.

16 В каких случаях надо опеределять КК, ОПК, КП, ОПП, деструктор?

КК по умолчанию создаётся компилятором. Но если объект содержит указатель на объекты в динамической памяти (куче), то его надо определять самому. По умолчанию компилятор будет копировать значение указателя, а не фактические данные, лежащие в куче. Это может привести к утечке памяти или неопределённому поведению программы.

Для ОПК ситуация аналогична КК. Определение КК и ОПК позволяют избежать ситуаций, когда два разных объекта будут ссылаться на одну и ту же память.

КП также создаётся компилятором по умолчанию. Но КП надо определять, если у вас определён КК. КП используется для копирования временных объектов (rvalue). Эти объекты удаляются из памяти сразу после выполнения операции, поэтому нет необходимости лишних действий копирования данных, и можно просто скопировать значения указателей. Если КК определён, а КП нет, то копилятор при копировании временного объекта вызовет КК.

Для ОПП ситуация аналогична КП. Если в классе определён ОПК, то надо определить и ОПП, а иначе компилятор при присвоении временного объекта вызовет ОПК и будут проделаны лишние действия.

Деструктор надо определять, если среди полей класса есть указатель. Надо правильно очистить данные из кучи, а иначе это будет приводить к утечке памяти.

Санкт-Петербургский государственный университет

Ответы

17 В каких случаях КК, ОПК, КП, ОПП, деструктор будут созданы компилятором?

Деструктор создаётся компилятором, если он не определён разработчиком и не удалён с использованием конструкции <сигнатура деструктора> = delete;

КК будет создан компилятором, если он не определён разработчиком и не удалён с использованием с delete. Если есть определённый КП или ОПП, то LongNumber x2(x1); выдаст ошибку компиляции и потребует определить и КК.

КП будет создан компилятором, если он не определён, он не удалён, не определён КК или ОПП.

ОПК будет создан компилятором, если он не определён, он не удалён, не определён КП или ОПП.

ОПП будет создан компилятором, если он не определён, он не удалён, не определён КП.

18 Что такое делегирующий конструктор?

Начиная с C++11 конструкторам разрешено вызывать другие конструкторы.

Delegate – передавать полномочия.

ДК — это конструктор, который вызывает другой конструктор.

Синтаксис вызова конструктора:

LongNumber(int n) : LongNumber() {...}

Запомнить!

Синтаксис делегирования конструктора не предотвращает случайное создание рекурсии конструктора: конструктор1 вызывает конструктор2, который вызывает конструктор1, и никаких ошибок не возникает, пока не произойдёт переполнения стека.

Санкт-Петербургский государственный университет

Ответы

19 Что такое наследование?

Наследование — это возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять свои.

```
class Derived : [virtual] [access-specifier] Base
{
   // member list
};
class Derived : [virtual] [access-specifier] Base1,
   [virtual] [access-specifier] Base2, . . .
{
   // member list
};
```

Ответы

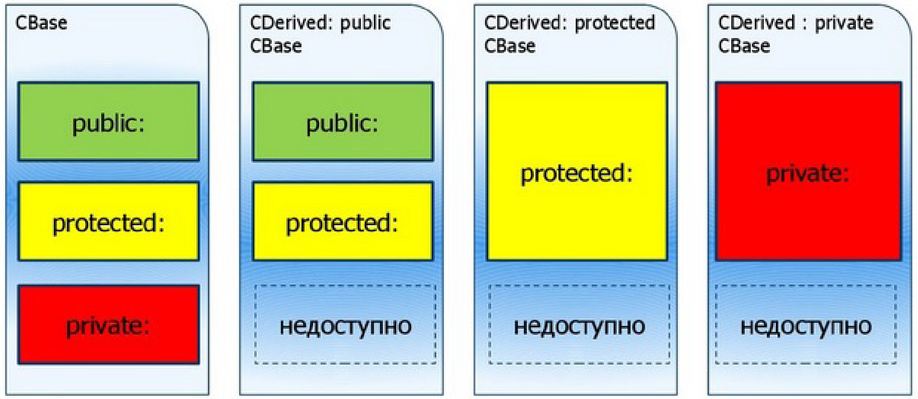
20 Какие бывают типы наследования?

Наследование бывает public, protected, private.

Наследование бывает одиночным (простым) / множественным.

Наследование бывает вирутальным / невиртуальным.

Спецификатор доспута класса контролирует разрешение производного класса на использование членов базового класса. Если спецификатор достуа опущен, то он считается private.



Санкт-Петербургский государственный университет

Ответы

21 Какие особенности наследования конструкторов, операторов присваивания?

Конструкторы и операторы присваивания не наследуются.

При этом, при вызове любого конструктора у производного класса (ПК) будет вызван КпоУ у базового класса (БК). Если в этом конструкторе есть делегирующий вызов какого-то конструктора БК, то КпоУ БК вызываться не будет.

Если БК определяет КсПа, то ПК не будет наследовать этот конструктор. Его надо определять.

Порядок вызова конструкторов:

1. Если в КПК явный вызов КБК отсутствует, то автоматически вызывается КБКпоУ.

2. Для иерархии из нескольких уровней К-ыБК-ов вызываются начиная с самого верхнего уровня (с самого базового).

3. В случае наследования от нескольких БК-ов их конструкторы вызываются в порядке объявления.

22 Какие особенности наследования деструктора?

Деструктор не наследуется.

При вызове деструктора производного класса (ДПК) автоматически будут вызваны все Д-ы БК-ов. Но в отличие от конструкторов, они будут вызваны в обратном порядке: сначала вызывается ДПК, затем деструкторы элементов класса, а потом ДБК.

Вызов деструкторов зависит от: типа переменной, вида связывания и определён ли деструктор БК как виртуальный или нет.

Санкт-Петербургский государственный университет

Ответы

23 Как запретить наследование класса?

Запретить наследование класса можно с помощью ключевого слова **final**.

```
1 class MyClass final {
2   …
3 }
```

24 Что такое полиморфизм? Какие механизмы связывания существуют в C++?

Полиморфизм — это возможность использовать в различных классах иерархии одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы.

Указателю на БК можно присвоить значение адреса объекта любого ПК. Вызов методов объекта происходит в соответствии с типом указателя, а не фактическим типом объекта (компилятор "думает наследованием").

Механизм раннего связывания — это механизм, при котором ссылки на методы объектов разрешаются во время компиляции. И этот механизм не относится к полиморфизму.

Механизм позднего связывания — это механизм, когда разрешение ссылок на метод происходит во время выполнения программы в зависимости от конкретного типа объекта, вызвавшего метод, а не типа его переменной. Этот механизм реализован с помощью виртуальных методов и является полиморфным.

Эти два механизма ещё называют статическим и динамическим связыванием.

Санкт-Петербургский государственный университет

Ответы

25 Что такое виртуальные функции и зачем они нужны?

Виртуальная функция — это функция-член, которую предполагается переопределить в производных классах.

Виртуальные функции обеспечивают вызов соответствующей функции для объекта. Они обеспечивают такой принцип ООП как полиморфизм. Позднее (динамическое) связывание виртуальной функции возможно только через указатель или ссылку, а не имя объекта.

Вирутальные функции в БК должны быть определены.

```
// Использование имени, а не указателя или ссылки
// Раннее (статическое) связывание
DerivedA dAs;
Base bs = dAs;
bs.print(); // Base::print()
bs.vm(); // Base::vm()

// Невиртуальный переопределённый метод print()
Base *dAb = new DerivedA();
dAb->print(); // Base::print()

// Виртуальный переопределённый метод vm()
// Позднее (динамическое) связывание. Полиморфизм
dAb->vm(); // DerivedA::vm()
```

Санкт-Петербургский государственный университет

Ответы

26 Для чего используют виртуальный деструктор?

Деструктор объекта вызывается по типу переменной указателя, а не фактическому типу объекта. Для этого деструктор БК должен быть виртуальным, а переопределённый деструктор в ПК должен быть описан с ключевым словом override (появилось в C++11).

```
1 Base dAb = DerivedA(); // ~Base()
2 DerivedA dA = DerivedA(); // ~DerivedA() -> ~Base()
3
4 DerivedA *dA_ptr = new DerivedA();
5 delete dA_ptr; // ~DerivedA() -> ~Base()
6
7 Base *dAb_ptr = new DerivedA();
8 delete dAb_ptr; // ~Base()
```

```
1 Base dAb = DerivedA(); // ~Base()
2 DerivedA dA = DerivedA(); // ~DerivedA() -> ~Base()
3
4 DerivedA *dA_ptr = new DerivedA();
5 delete dA_ptr; // ~DerivedA() -> ~Base()
6
7 Base *dAb_ptr = new DerivedA();
8 delete dAb_ptr; // OK ~DerivedA() -> ~Base()
```

27 Что такое абстрактный класс и чистая виртуальная функция?

Абстрактный класс (АК) предназначен для представления общих понятий, которые предполагается конкретизировать в производных классах. АК может использоваться только в качестве БК для других классов. Объекты АК создавать нельзя.

С точки зрения синтаксиса C++ АК — это класс, который содержит хотя бы один чисто виртуальный метод.

Чистая виртуальная фукнция — это виртаульная функция, не имеющая релизации. Её синтаксис объявления:

virtual void func() = 0;

Санкт-Петербургский государственный университет

Ответы

28 Может ли иметь реализацию pure virtual function?

Можно определять чистые виртуальные функции (ЧВФ), у которых есть тело. При реализации тела для ЧВФ тело должно быть предоставлено отдельно от объявления класса (не встроенным). При этом сам класс будет считаться АК, и все ПК должны будут реализовать эту ЧВФ. Определение ЧВФ в БК может рассматриваться как определение по умолчанию и вызываться через имя БК: «БК::ЧВФ();»

```cpp
class A {
    public:
        virtual void func() = 0;
};

void A::func() {
    // Код
}
```

```cpp
class A {
    public:
        virtual void func() = 0 {}; // Ошибка компиляции
};
```

Ответы

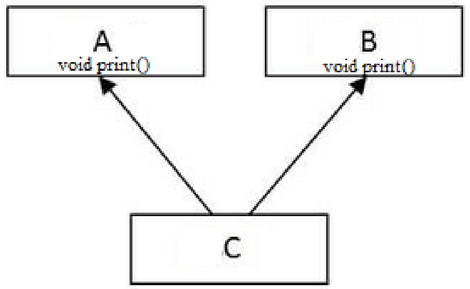
29 Что такое множественное наследование? Какие у него проблемы?

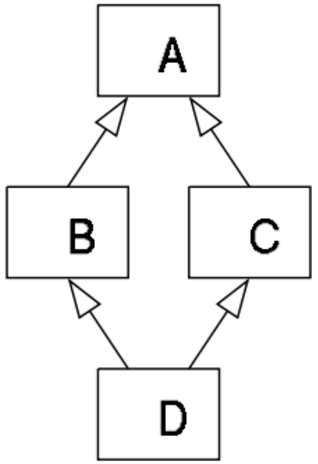
В C++ в отличие от других языков есть возможность наследовать класс от нескольких классов. К сожалению это приводит к проблемам и таких ситуаций стараются избегать.

Проблемы:

1. Как вызвать нужную реализацию функции, которая есть сразу у нескольких БК?

2. Проблема ромба. Ромбовидное наследование возникает, когда два или несколько БК наследованы от одного и тоге же класса. В этом случае наш класс наследник будет иметь две копии БК самого верхнего уровня.





Санкт-Петербургский государственный университет

Ответы

30 Как решить проблему ромбовидного наследования?

Чтобы решить проблему ромбовидного наследования, надо применить виртуальное наследование к наследуемым классам, имеющим общий БК. Это будет означать, что будет создан только один базовый объект. И за его создание будет отвечать конструктор ПК, минуя конструкторы виртуальных БК-ов.

А еще лучше не использовать множественное наследование, и в частности ромбовидный случай. А заменить множественное наследование композаицией или изменить архитектуру классов.

```
Class A {};
Class B: public A {};
Class C: public A {};
Class D: virtual public B, virtual public C {};
// Или
Class D {
    B b;
    C c;
};
```

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

31

```cpp
class A {
    public:
        A() { std::cout << "A" << std::endl; }
        ~A() { std::cout << "~A" << std::endl; }
};
class B: public A {
    public:
        B() { std::cout << "B" << std::endl; }
        ~B() { std::cout << "~B" << std::endl; }
};
B x;
```

Результат: AB~B~A

Статическое связывание происходит по типу переменной.
Тип переменной – это ПК.
Конструкторы при наследовании выполняются в порядке от верхнего БК до нижнего ПК.
Деструкторы выполняются в обратном порядке.

32

```cpp
class A {
    public:
        A() {}
        ~A() {}
        void print() { std::cout << "A::print" << std::endl; }
};
class B: protected A {
    public:
        B() {}
        ~B() {}
        using A::print;
};
B x;
x.print();
```

Результат: A::print

Статическое связывание происходит по типу переменной.
B наследуется от A с видимостью protected – значит методы класса A не могут вызываться у объекта класса B. Но конструкция using A::print в секции public класса B изменяет область видимости метод print класса A для объектов ПК B.

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

33

```cpp
class A {
    public:
        A() { std::cout << "A" << std::endl; }
        A(const A&) = delete;
        A(A&&) { std::cout << "A&&" << std::endl; }
        ~A() { std::cout << "~A" << std::endl; }
};
class B: public A {
    public:
        B() { std::cout << "B" << std::endl; }
        B(const B&) { std::cout << "const B&" << std::endl; }
        B(B&&) { std::cout << "B&&" << std::endl; }
        ~B() { std::cout << "~B" << std::endl; }
};
B xb;
A xa(xb);
```

Результат: Ошибка компиляции

Строка 15 имеет статическое связывание и должно было бы вывестись AB.
Но в строке 16 происходит вызов конструктора копирования.
Т.к. тип переменной xa – это тип БК, то xb – будет преобразовано к A и будет вызван КК класса A. Но КК у A удалён с помощью ключевого слова delete. Поэтому будет ошибка компиляции.

34

```cpp
class A {
    public:
        A() { std::cout << "A" << std::endl; }
        A(const A&) { std::cout << "const A&" << std::endl; }
        A(A&&) { std::cout << "A&&" << std::endl; }
        ~A() { std::cout << "~A" << std::endl; }
};
class B: public A {
    public:
        B() { std::cout << "B" << std::endl; }
        B(const B&) { std::cout << "const B&" << std::endl; }
        B(B&&) { std::cout << "B&&" << std::endl; }
        ~B() { std::cout << "~B" << std::endl; }
};
A x = B();
```

Результат: ABA&&~B~A~A

= при объявлении переменной обозначает инициализацию, а не присвоение.
Но вначале будет выполнено выражение справа – КпоУ класса B → AB.
Т.к. справа rvalue, то будет выполняться КП.
Т.к. A – БК, а B – ПК, то B будет преобразован в A и будет вызван КП класса A → A&&.
Сразу после инициализации будет вызван деструктор для rvalue → ~B~A.
При выходе из области видимости будет вызван деструктор x → ~A.

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

35

```
1 class A {
2     public:
3         A() { std::cout << "A" << std::endl; }
4         ~A() { std::cout << "~A" << std::endl; }
5 };
6 class B: public A {
7     public:
8         B() { std::cout << "B" << std::endl; }
9         ~B() { std::cout << "~B" << std::endl; }
10 };
11 B* xb = new B();
12 delete xb;
13 A* xa = new B();
14 delete xa;
```

Результат: AB~B~AAB~A и утечка памяти

Конструкторы вызываются по фактически создаваемому объекту всегда. А вызов деструкторов будет зависеть от типа переменной, вида связывания и наличия virtual у деструктора БК.
Строка 11. AB
Строка 12. ~B~A – тип переменной B (сам фактический объект)
Строка 13. AB
Строка 14. ~A – тип переменной A, связывание динамическое, но деструктор в БК не virtual

36

```
1 class A {
2     public:
3         A() { std::cout << "A" << std::endl; }
4         virtual ~A() { std::cout << "~A" << std::endl; }
5 };
6 class B: public A {
7     public:
8         B() { std::cout << "B" << std::endl; }
9         ~B() { std::cout << "~B" << std::endl; }
10 };
11 A* xa = new B();
12 delete xa;
```

Результат: AB~B~A

Конструкторы вызываются по фактически создаваемому объекту всегда. А вызов деструкторов будет зависеть от типа переменной, вида связывания и наличия virtual у деструктора БК.
Строка 11. AB
Строка 12. ~B~A – тип переменной A, связывание динамическое, а деструктор в БК virtual! И компилятор в итоге думает полиморфизмом.

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

37

```cpp
class A {
    public:
        A() {}
        virtual ~A() {}
        void print() { std::cout << "A::print" << std::endl; }
        virtual void vm() { std::cout << "A::vm" << std::endl; }
};
class B: public A {
    public:
        B() {}
        ~B() {}
        void print() { std::cout << "B::print" << std::endl; }
        void vm() { std::cout << "B::vm" << std::endl; }
};
A* xa = new B();
xa->print();
xa->vm();
```

Результат: A::print и B::vm

Когда у нас тип переменной является БК, а не фактическим типом объекта ПК, то при динамическом связывании при вызове переопределённых методов важно понять "Чем будет думать компилятор?".
Наследованием или полиморфизмом?
Если метод объявлен в БК без virtual, то будет выполняться наследование – и выберется наследуемая реализация, т.е. БК.
Если с virtual, то – полиморфизм – и выбирается реализация по фактическому типу объекта.

38

```cpp
class A {
    public:
        A() { std::cout << "A" << std::endl; }
        virtual ~A() { std::cout << "~A" << std::endl; }
        virtual void print() = 0;
};
class B: public A {
    public:
        B() { std::cout << "B" << std::endl; }
        ~B() { std::cout << "~B" << std::endl; }
        void print() { std::cout << "B::print" << std::endl; };
};
A xa;
```

Результат: Ошибка компиляции

В строке 13 инициализируется КпоУ объект типа A. Но у класса A есть чистый виртуальный метод, а значит класс A является абстрактным классом. Объекты абстрактных классов создавать нельзя.

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

39

```
1 class A {
2     public:
3         A() { std::cout << "A" << std::endl; }
4         virtual void print() { std::cout << "A::print" << std::endl; }
5 };
6 class B: public A {
7     public:
8         B() { std::cout << "B" << std::endl; print(); }
9 };
10 class C: public B {
11     public:
12         C() { std::cout << "C" << std::endl; }
13         void print() { std::cout << "C::print" << std::endl; };
14 };
15 C x;
```

Результат: A B A::print C

Таблица виртуальных функций создаётся по каждому классу во время вызова его конструктора. На момент вызова print() в строке 8 у нас произошёл вызов конструктора A и конструктора B, а конструктор C ещё не был вызван. И поэтому компилятор не знает реализации print в C.
Несмотря на то, что у нас статическое связывание, но рассуждения аналогичные. Компилятор привязывает таблицу виртуальных функций к конструкторам. И будет выбрана та реализация, которая по иерархии находится либо в этом же классе, либо в БК.

40

```
1 class A {
2     public:
3         void print() { std::cout << "A::print" << std::endl; };
4 };
5 class B {
6     public:
7         void print() { std::cout << "B::print" << std::endl; };
8 };
9 class C: public A, public B {};
10 C x;
11 x.print();
```

Результат: Ошибка компиляции

Первая проблема множественного наследования "ambiguous" — двусмысленный выбор.

Санкт-Петербургский государственный университет

Какой результат программы? Объясните

41

```
1 class A {
2     int value[10];
3 };
4 class B: public A {};
5 class C: public A {};
6 class D: public B, public C {};
7 std::cout << sizeof(D) << std::endl;
```

Результат: 80

Проблема наследования "Ромб".
Т.к. БК B и C наследованы от одного и того же A, то класс D будет содержать две копии данных класса A.

42

```
1 class A {
2     public:
3         int value;
4         A() : value(10) {};
5 };
6 class B: public A {};
7 class C: public A {};
8 class D: public B, public C {};
9 D d;
10 d.B::value = 1;
11 std::cout << d.C::value << std::endl;
```

Результат: 10

Проблема наследования "Ромб".
Т.к. БК B и C наследованы от одного и того же A, то класс D будет содержать две копии данных класса A. Одна копия будет относиться к B, а другая к C.
d.B::value = 1 – меняется значение копии данных A, относящейся к B, а у копии C останется значение из КпоУ.

Санкт-Петербургский государственный университет

Задачи для практики на рекурсию

1. Найти количество и вывести на экран все n-значные числа, у которых сумма цифр в m раз меньше их произведения.
2. Найти количество и вывести на экран все n-значны полиндромы, у которых сумма цифр делится на d.
3. Написать быструю сортировку.
4. Найти количество и вывести все варианты разложения числа n на слагаемые. Разложение отличающееся перестановкой не учитывается.
5. Найти количество и вывести все варианты рзложения числа n на k слагае-мых. Разложение отличающееся перестановкой не учитывается.

Санкт-Петербургский государственный университет

Задачи для практики на рекурсию

1

```cpp
// Найти количество и вывести на экран все n-значные числа,
// у которых сумма цифр в m раз меньше их произведения.

void count_numbers(
    const int n,
    const int m,
    std::string number,
    int& count
) {
    const char zero = '0';
    if (number.size() == n) {
        int summ = 0;
        int multiply = 1;
        for (int i = 0; i < n; i++) {
            summ += number[i] - zero;
            multiply *= number[i] - zero;
        }
        if (multiply / summ == m && multiply % summ == 0) {
            count++;
            std::cout << number << std::endl;
        }
        return;
    }

    const int digits_count = 10;
    for (int i = 0; i < digits_count; i++) {
        if (number.size() == 0 && i == 0) {
            continue;
        }
        number += char(i + zero);
        count_numbers(n, m, number, count);
        number.pop_back();
    }
}

// Вызов процедуры
const int n = 4;
const int m = 10;
std::string number;
int count = 0;
count_numbers(n, m, number, count);
std::cout << "Количество чисел = " << count << std::endl;
```

2

```cpp
// Найти количество и вывести на экран все n-значные
// полиндромы, у которых сумма цифр делится на d.

void count_palindromes(
    const int n,
    const int d,
    std::string number,
    int& count
) {
    const char zero = '0';
    const int size = number.size();
    if (size == n) {
        int summ = 0;
        for (int i = 0; i < n; i++) {
            summ += number[i] - zero;
        }
        if (summ % d == 0) {
            count++;
            std::cout << number << std::endl;
        }
        return;
    }

    const int digits_count = 10;
    for (int i = 0; i < digits_count; i++) {
        if (size == 0 && i == 0) {
            continue;
        }
        if (size >= n / 2 && int(number[n - size - 1]) != i + '0') {
            continue;
        }
        number += char(i + zero);
        count_palindromes(n, d, number, count);
        number.pop_back();
    }
}

// Вызов процедуры
const int n = 4;
const int d = 7;
std::string number;
int count = 0;
count_palindromes(n, d, number, count);
std::cout << "Количество полиндромов = " << count << std::endl;
```

Санкт-Петербургский государственный университет

Задачи для практики на рекурсию

3

```cpp
// Написать быструю сортировку.

namespace IBusko {
    void quick_sort(int arr[], int l, int r);
    void swap(int& x, int& y);

    void quick_sort(int arr[], int l, int r) {
        if (l >= r) return;

        int x = arr[(l + r) / 2];
        int i = l;
        int j = r;
        while (i <= j) {
            while (arr[i] < x) {
                i++;
            }
            while (arr[j] > x) {
                j--;
            }
            if (i <= j) {
                swap(arr[i++], arr[j--]);
            }
        }

        quick_sort(arr, l, j);
        quick_sort(arr, i, r);
    }

    void swap(int& x, int& y) {
        int temp = x;
        x = y;
        y = temp;
    }
}

// Вызов процедуры
int arr[] = { 3, 12, 4, 0, 6, -2, -6, 11, 3, 5, 8 };
const int size = sizeof(arr) / sizeof(arr[0]);
IBusko::quick_sort(arr, 0, size - 1);
for(int i = 0; i < size; i++) {
    std::cout << arr[i] << ' ';
}
```

4

```cpp
// Найти количество и вывести все варианты разложения числа n
// на слагаемые. Разложение отличающееся перестановкой не учитывается.

void decomposite_in_terms(
    const int n,
    std::vector<int> vec,
    int summ_rest,
    int& count
) {
    if (summ_rest == 0) {
        count++;
        std::cout << n << " = ";
        for (int i = 0; i < vec.size() - 1; i++) {
            std::cout << vec[i] << " + ";
        }
        std::cout << vec[vec.size() - 1] << std::endl;
        return;
    }

    for (int i = 1; i <= summ_rest; i++) {
        if (vec.empty()) {
            vec.push_back(i);
            decomposite_in_terms(n, vec, summ_rest - i, count);
            vec.clear();
        } else if (vec[vec.size() - 1] >= i) {
            vec.push_back(i);
            decomposite_in_terms(n, vec, summ_rest - i, count);
            vec.pop_back();
        }
    }
}

// Вызов процедуры
std::vector<int> terms;
const int n = 10;
int count = 0;
std::cout
    << "Разложение числа "
    << n
    << " на слагаемые имеет варианты: "
    << std::endl;
decomposite_in_terms(n, terms, n, count);
std::cout << "Количество вариантов = " << count << std::endl;
```

Санкт-Петербургский государственный университет

Задачи для практики на рекурсию

5

```cpp
// Найти количество и вывести все варианты рзложения числа n
// на k слагаемых. Разложение отличающееся перестановкой не учитывается.

void decomposite_in_k_terms(
    const int n,
    const int k,
    std::vector<int> vec,
    int summ_rest,
    int& count
) {
    if (vec.size() > k || (summ_rest == 0 && vec.size() < k)) return;

    if (summ_rest == 0 && vec.size() == k) {
        count++;
        std::cout << n << " = ";
        for (int i = 0; i < vec.size() - 1; i++) {
            std::cout << vec[i] << " + ";
        }
        std::cout << vec[vec.size() - 1] << std::endl;
        return;
    }

    for (int i = 1; i <= summ_rest; i++) {
        if (vec.empty()) {
            vec.push_back(i);
            decomposite_in_k_terms(n, k, vec, summ_rest - i, count);
            vec.clear();
        } else if (vec[vec.size() - 1] >= i) {
            vec.push_back(i);
            decomposite_in_k_terms(n, k, vec, summ_rest - i, count);
            vec.pop_back();
        }
    }
}

// Вызов процедуры
std::vector<int> terms;
const int n = 10;
const int k = 3;
int count = 0;
std::cout
    << "Разложение числа "
    << n
    << " на "
    << k
    << " слагаемых имеет варианты: "
    << std::endl;
decomposite_in_k_terms(n, k, terms, n, count);
std::cout << "Количество вариантов = " << count << std::endl;
```

Санкт-Петербургский государственный университет

Дополнительные задания

1

```cpp
#include <cstring>

LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber::LongNumber(const char* const str) {
    int str_length = std::strlen(str);
    if (str[0] == '-') {
        sign = -1;
        length = str_length - 1;
    } else {
        sign = 1;
        length = str_length;
    }

    numbers = new int[length];
    for (int i = 0; i < length; i++) {
        numbers[i] = str[str_length - i - 1] - '0';
    }
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```

2

```cpp
LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber::LongNumber(const LongNumber& x) {
    length = x.length;
    sign = x.sign;
    numbers = new int[length];
    for (int i = 0; i < length; i++) {
        numbers[i] = x.numbers[i];
    }
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```

Санкт-Петербургский государственный университет

Дополнительные задания

3

```
LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber::LongNumber(LongNumber&& x) {
    length = x.length;
    sign = x.sign;
    numbers = x.numbers;
    x.numbers = nullptr;
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```

4

```
#include <cstring>

LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber& LongNumber::operator = (const char* const str) {
    int str_length = std::strlen(str);
    if (str[0] == '-') {
        sign = -1;
        length = str_length - 1;
    } else {
        sign = 1;
        length = str_length;
    }

    delete [] numbers;
    numbers = new int[length];
    for (int i = 0; i < length; i++) {
        numbers[i] = str[str_length - i - 1] - '0';
    }

    return *this;
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```

Санкт-Петербургский государственный университет

Дополнительные задания

5

```cpp
LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber& LongNumber::operator = (const LongNumber& x) {
    if (this == &x) return *this;

    length = x.length;
    sign = x.sign;

    delete [] numbers;
    numbers = new int[length];
    for (int i = 0; i < length; i++) {
        numbers[i] = x.numbers[i];
    }

    return *this;
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```

6

```cpp
LongNumber::LongNumber() : length(1), sign(1) {
    numbers = new int[length];
    numbers[0] = 0;
}

LongNumber& LongNumber::operator = (LongNumber&& x) {
    length = x.length;
    sign = x.sign;

    delete [] numbers;
    numbers = x.numbers;
    x.numbers = nullptr;

    return *this;
}

LongNumber::~LongNumber() {
    length = 0;
    delete [] numbers;
    numbers = nullptr;
}
```