Лабораторная работа 2. ООП. Классы, конструктуры, перегрузка операций

1. ЦЕЛЬ РАБОТЫ

Повторить основные понятия ООП: «объект», «класс», «инкапсуляция»; познакомиться со способами описания классов и объектов в языке C++; познакомиться с возможностью перегрузки операторов и использования различных конструкторов объектов класса; разработать приложения по своим вариантам заданий.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Класс – это набор методов и свойств, описывающих поведение объекта. *Объект* – это конкретный представитель определенного класса. Таким образом, класс подобен стандартному элементу управлению в визуальных средах разработки, но без графического интерфейса. Класс используется аналогично, но его нельзя программировать визуально, как это делается в случае элементов управления.

Программу, в которой созданные классы близко совпадают с понятиями прикладной задачи, обычно легче понять, модифицировать и отладить, чем программу, в которой этого нет.

Инкапсуляция является одним из основных понятий ООП. Инкапсуляция – объединение в одном месте описания всех методов и данных класса. Инкапсуляция облегчает понимание работы программы, а также ее отладку и модификацию, так как внутренняя реализация используемых объектов редко интересует разработчика программного проекта; главное, чтобы объект обеспечивал функции, которые он должен предоставить. Кроме того, инкапсуляция защищает данные и методы класса от внешнего вмешательства или неправильного использования. Другими словами, инкапсуляция означает сокрытие деталей реализации класса внутри него самого.

Взаимодействие с объектом происходит через интерфейс, а детали реализации остаются инкапсулированными. В объектах интерфейсами являются свойства, методы и события. Только они предоставляются данным объектом в распоряжение других объектов. Таким образом, инкапсуляция обеспечивает использование объекта, не зная, как он реализован внутри.

Перегрузка операций

Язык C++ позволяет переопределять для классов существующие обозначения операций. Это называется *перегрузкой операций.* Благодаря ей класс можно сделать таким, что он будет вести себя подобно встроенному типу. Например, в классе можно перегрузить такие операци: `+, -, *, /, =, >, <, >=, <=, +=` и другие.

Функции-операции, реализующие перегрузку операций, имеют вид

```
тип_возвращаемого_значения operator знак_операции
([операнды]) {тело функции}
```

Если функция является элементом класса, то первый операнд соответствующей операции будет самим объектом, для которого вызвана функция-

операция (его не надо указывать). В случае одноместной операции список параметров будет пуст. Для двухместных операций функция будет иметь один параметр, соответствующий второму операнду. Если функция-операция не является элементом класса, она будет иметь один параметр в случае одноместной операции и два — в случае двухместной.

Для перегрузки операций существуют такие правила:

- Приоритет и правила ассоциации для перегруженных операций остаются теми же самыми, что и для операций над встроенными типами.
- Нельзя изменить поведение операции по отношению к встроенному типу.
- Функция-операция должна быть либо элементом класса, либо иметь один или несколько параметров типа класса.
- Функция-операция не может иметь аргументов по умолчанию.
- Обычно операцию присваивания определяют так, чтобы она возвращала ссылку на свой объект. В этом случае сохраняется семантика арифметических присваивании, допускающая последовательные присваивания в выражении (т.е. `c = b = a;`).
- Невозможно изменить синтаксис перегруженных операций. Одноместные операции должны быть одноместными, а двухместные — двухместными.
- Нельзя изобретать новые обозначения операций. Возможные операции ограничиваются тем списком, что приведен в начале этого раздела.
- Желательно сохранять смысл перегружаемой операции.

Конструкторы и деструкторы. Как известно, в классе могут быть объявлены две специальные функции-элемента – *конструктор* и *деструктор.*

Конструктор отвечает за создание представителей данного класса. Его объявление записывается без типа возвращаемого значения и ничего не возвращает, а имя должно совпадать с именем класса. Конструктор может иметь любые параметры, необходимые для *конструирования,* т. е. создания, нового представителя класса. Если конструктор не определен, компилятор генерирует *конструктор по умолчанию,* который просто выделяет память, необходимую для размещения представителя класса.

С помощью конструктора можно при создании объекта элементам данных класса присвоить некоторые начальные значения, определяемые в программе. Это реализуется путем передачи аргументов конструктору объекта.

Конструкторы можно перегружать, чтобы обеспечить различные варианты задания начальных значений объектов класса. Конструктор может содержать значения аргументов по умолчанию. Для каждого класса может существовать только один конструктор с умолчанием.

Когда объявляется объект класса, между именем объекта и точкой с запятой в скобках можно указать *список инициализации элементов*. Эти начальные значения передаются в конструктор класса при вызове конструктора.

Обычно различают следующие виды конструкторов:

- конструктор без аргументов (конструктор по умолчанию);

- конструктор инициализации;

- конструктор копирования;

- конструктор перемещения и др.

Если для класса не определено никакого конструктора, компилятор создает сам конструктор с умолчанием. Такой конструктор не задает никаких начальных значений, так что после создания объекта нет никакой гарантии, что он находится в непротиворечивом состоянии.

Конструктор, не требующий аргументов, называется *конструктором по умолчанию*. Конструктор по умолчанию не имеет аргументов и инициализирует все переменные члены какими-либо начальными значениями. Если программист явно указал только конструктор с параметрами, то компилятор не будет создавать конструктор по умолчанию.

Конструктор копирования предназначен для инициализации объекта путем копирования значений членов из объекта того же типа. Если члены класса являются простыми типами, такими как скалярные значения, конструктор копирования, созданный компилятором, достаточно, и вам не нужно определять собственные. Если для класса требуется более сложная инициализация, необходимо реализовать пользовательский конструктор копий. Например, если член класса является указателем, необходимо определить конструктор копий, чтобы выделить новую память и скопировать значения из другого объекта, на который существует указатель. Созданный компилятором конструктор копий просто копирует указатель, так что новый указатель по-прежнему указывает на прежнее расположение в памяти.

При определении конструктора копий необходимо также определить оператор присваивания копированием (=).

Конструктор перемещения — это специальная функция-член, передающая право собственности на данные существующего объекта новой переменной без копирования исходных данных. Он принимает ссылку rvalue в качестве первого параметра, а все последующие параметры должны иметь значения по умолчанию. Конструкторы перемещения могут значительно повысить эффективность программы при передаче больших объектов. Компилятор выбирает конструктор перемещения, когда объект инициализирован другим объектом того же типа, если другой объект будет уничтожен и больше не нуждается в его ресурсах.

Деструктор отвечает за уничтожение представителей класса. Если деструктор не определен, генерируется деструктор по умолчанию, который просто возвращает системе занимаемую объектом память. Деструктор объявляется без типа возвращаемого значения, ничего не возвращает и не имеет параметров. Имя деструктора совпадает с именем класса, но перед ним ставится символ ~ (тильда). Класс может иметь только один деструктор, то есть перегрузка деструктора запрещена. Деструкторы имеют смысл в классах, использующих динамическое распределение памяти под объекты (например, для массивов или списков, …).

Области памяти, занятые данными базовых типов, таких как `int`, `float`, `double` и др., выделяются и освобождаются автоматически и не нуждаются в помощи конструктора и деструктора.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1. Ознакомьтесь с теоретическим материалом, приведенным в пункте «Краткие теоретические положения» данных методических указаний, конспектом лекций по данной теме и рекомендуемой литературой.

Задание 2. Разработайте программу по своему варианту. В вариантах заданий указаны примерные требования к функционалу класса. **Согласуйте точное задание по своему варианту перед тем, как начать выполнять реализацию класса.**

Покажите результаты работы преподавателю. Оформите отчет по работе.

4. ОФОРМЛЕНИЕ ОТЧЕТА

Отчет по работе должен содержать:

- название и цель работы;
- номер варианта;
- файлы *.h и *.cpp, содержащие описание и реализацию классов;
- текст основной программы с комментариями;
- описание разработанных классов, структур данных и функций;
- результаты работы программы для нескольких вариантов входных данных.

5. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шилдт Г. C++: базовый курс, 3-е издание. : Пер. с англ. – М.: «Издательский дом «Вильямс», 2005. – 624 с.

2. Конструкторы (C++) https://learn.microsoft.com/ru-ru/cpp/cpp/constructors-cpp?view=msvc-170

3. Классы. https://academy.yandex.ru/handbook/cpp/article/classes

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие типы данных называются абстрактными в C++?
2. Каким образом производиться объявление класса?
3. Какие спецификаторы доступа используются при объявлении класса?
4. В чем заключается инкапсуляция? Как реализуется инкапсуляция при создании абстрактных типов данных?
5. Какие правила существуют для перегрузки операций для классов?
6. Что такое полиморфизм? Как реализуется полиморфизм при создании абстрактных типов данных?

ВАРИАНТЫ ЗАДАНИЙ

1. Создать класс `BitString` для работы с битовыми строками не более чем из 100 бит. Битовая строка должна быть представлена массивом типа `unsigned char`, каждый элемент которого принимает значение 0 или 1. Реальный размер массива задается как аргумент конструктора инициализации. Должны быть реализованы все традиционные операции для работы с битовыми строками: `and`, `or`, `xor`, `not`. Реализовать сдвиг влево и сдвиг вправо на заданное количество битов.
2. Создать класс `Decimal` для работы с беззнаковыми целыми десятичными числами, используя для представления числа массив из 100 элементов типа `unsigned char`, каждый из которых является десятичной цифрой. Младшая цифра имеет меньший индекс (единицы — в нулевом элементе массива). Реальный размер массива задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых в C++, и операции сравнения.
3. Создать класс `Hex` для работы с беззнаковыми целыми шестнадцатеричными числами, используя для представления числа массив из 100 элементов типа `unsigned char`, каждый из которых является шестнадцатеричной цифрой. Младшая цифра имеет меньший индекс. Реальный размер массива задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых в C++, и операции сравнения.

5. Создать класс `Polinom` для работы с многочленами до 100-й степени. Коэффициенты должны быть представлены массивом из 100 элементов-коэффициентов. Младшая степень имеет меньший индекс (нулевая степень — нулевой индекс). Размер массива задается как аргумент конструктора инициализации. Реализовать арифметические операции и операции сравнения, вычисление значения полинома для заданного значения x, дифференцирование, интегрирование.
6. Создать класс `Fraction` для работы с беззнаковыми дробными десятичными числами. Число должно быть представлено двумя массивами типа `unsigned char`: целая и дробная часть, каждый элемент — десятичная цифра. Для целой части младшая цифра имеет меньший индекс, для дробной части старшая цифра имеет меньший индекс (десятые — в нулевом элементе, сотые — в первом, и т. д.). Реальный размер массивов задается как аргумент конструктора инициализации. Реализовать арифметические операции сложения, вычитания и умножения, и операции сравнения.

7. Класс `List` (однонаправленный список).

Класс	Элементы данных*	Интерфейс
Структура `Node`	Значение, указатель на следующий элемент	Конструктор
`List`	Корень (голова) списка,	Конструкторы, методы для добавления узла в конец списка, добавления узла в начало списка,

	Количество узлов списка	вставки узла после указанного узла, удаления узла, получения количества узлов, проверки, пуст ли список, подсчет числа вхождений значения в список, операции =, + (получения одного списка из двух), ==, !=, <<, >>.

Разработать и отладить программу с примерами создания и использования объектов класса `List`.

* Один из вариантов, по согласованию с преподавателем можно реализовать по-другому.

8. Класс `List` (двунаправленный список). Описание класса:

Класс	Элементы данных*	Интерфейс
Структура `Node`	Значение, указатель на следующий элемент, указатель на предыдущий	Конструктор
`List`	Голова и хвост списка, Количество узлов списка	Конструкторы, методы для добавления узла в конец списка, вставки узла после указанного узла, вставки в начало списка, вставки в конец списка, удаления узла, удаления последнего узла, получения количества узлов, проверки, пуст ли список, операции =, + (получения одного списка из двух), ==, !=, <<, >>.

9. Класс "Матрица". Класс должен иметь следующие переменные:

- двумерный массив вещественных чисел;
- количество строк и столбцов в матрице.

Класс должен иметь следующие методы:

- сложение с другой матрицей;
- умножение на число;
- вывод на печать;
- умножение матриц.

10. Класс `CString` (строка). Описание класса:

Класс	Элементы данных	Интерфейс
`CString`	Длина строки, указатель на буфер	Конструкторы, функция `Len` (длина строки), операции =, ==, !=, <, > (сравнение строк), + , <<, >>

	динамически выделенной памяти для размещения строки	

Разработать и отладить программу с примерами создания и использования объектов класса `CString`. Добавить в программу возможность создания и сортировки (упорядочения) массива строк.

11. Класс `Arr` для работы с одномерным массивом целых чисел.

Класс	Элементы данных	Интерфейс
`Arr`	Размер массива, указатель на буфер динамически выделенной памяти для размещения элементов массива	Конструкторы, функция `Sort` (упорядочение массива), `Add` (добавление элемента в массив), `Del` (удаление элемента из массива), операции =, + (слияние массивов), * (умножение на скаляр), <<, >>

Разработать и отладить программу с примерами создания и использования объектов класса `Arr`.

12. Класс Stack (на массиве) для хранения целых чисел

Класс	Элементы данных	Интерфейс
Stack	Текущий размер стека, максимальный размер стека, массив для хранения стека	Конструкторы, push (поместить в стек), pop (извлечь из стека), top (получить верхний элемент), операции =, + (слияние двух стеков), <<, >>

13. Класс Stack (на связном списке) для хранения символов

Класс	Элементы данных	Интерфейс
Stack	Текущий размер стека, указатели на голову и хвост списка	Конструкторы, push (поместить в стек), pop (извлечь из стека), top (получить верхний элемент), операции =, + (слияние двух стеков), <<, >>

Также попробовать использовать разработанный стек для решения задачи о правильности расстановки скобок.

14. Класс Queue (на массиве) для хранения целых чисел

Класс	Элементы данных	Интерфейс

Queue	Текущий размер очереди, максимальный размер очереди, массив для хранения очереди	Конструкторы, push (поместить в очередь), pop (извлечь из очереди), front (получить первый элемент), back (получить последний элемент очереди), операции =, + (слияние двух очередей), <<, >>

15. Класс Queue (на связном списке) для хранения строк

Класс	Элементы данных	Интерфейс
Queue	Текущий размер очереди, указатели на голову и хвост очереди	Конструкторы, push (поместить в очередь), pop (извлечь из очереди), front (получить первый элемент), back (получить последний элемент очереди), операции =, + (слияние двух очередей), <<, >>

16. Класс Deque (дек – двухсторонняя очередь) для хранения целых чисел (на массиве)

Класс	Элементы данных	Интерфейс
Queue	Текущий размер очереди, максимальный размер очереди, массив для хранения очереди	Конструкторы, push_back(), push_front (), pop_back(), pop_front(), front (), back (), операции =, + (слияние двух очередей), <<, >>

17. Класс Deque (дек – двухсторонняя очередь) для хранения целых чисел (на связном списке) для хранения символов

Класс	Элементы данных	Интерфейс
Queue	Текущий размер очереди, указатели на голову и хвост очереди	Конструкторы, push_back(), push_front (), pop_back(), pop_front(), front (), back (), операции =, + (слияние двух очередей), <<, >>

18. Класс BinSearchTree - Бинарное дерево поиска для хранения строк

Класс	Элементы данных	Интерфейс
Структура `Node`	Значение данных, указатели на левого и правого потомка	Конструктор
BinSe-archTree	указатель на корень дерева	Конструкторы, insert() – вставка элемента в дерево, find() – проверка, есть ли значение в

		дереве, операции =, + (слияние двух деревьев), <<, >>

19. Класс BinSearchTree - Бинарное дерево поиска для хранения целых чисел

Класс	Элементы данных	Интерфейс
Структура `Node`	Значение данных, указатели на левого и правого потомка	Конструктор
BinSearchTree	указатель на корень дерева	Конструкторы, insert() – вставка элемента в дерево, find() – проверка, есть ли значение в дереве, delete() – удаление узла с заданным значением данных, операции =, <<, >>

20. Класс BinSearchTree - Бинарное дерево поиска для хранения целых чисел

Класс	Элементы данных	Интерфейс
Структура `Node`	Значение данных, указатели на левого и правого потомка	Конструктор
BinSearchTree	указатель на корень дерева	Конструкторы, insert() – вставка элемента в дерево, findX() – определение длины пути от корня дерева до ближайшей вершины с элементом X, delete() – удаления узла, имеющего одно сына, операции =, <<, >>

21. Класс BinSearchTree - Бинарное дерево поиска для хранения целых чисел

Класс	Элементы данных	Интерфейс
Структура `Node`	Значение данных, указатели на левого и правого потомка	Конструктор
BinSearchTree	указатель на корень дерева	Конструкторы, insert() – вставка элемента в дерево, count() – подсчета числа вхождений заданного элемента в дерево, length() – определение высоты дерева, операции =, <<, >>

20. Создать класс `String` для работы со строками, аналогичными строкам Turbo Pascal (строка представляется как массив 255 байт, длина – в первом байте). Максимальный размер строки должен задаваться. Обязательно должны быть реализованы: определение длины строки, поиск подстроки в строке, удаление подстроки из строки, вставка подстроки в строку, сцепление двух строк.

23. Реализовать класс Set (множество) не более чем из 256 элементов-символов. Решение должно обеспечивать включение элемента в множество, исключение элемента из множества, объединение, пересечение и разность множеств, вычисление количества элементов в множестве, проверку присутствия элемента в множестве, проверку включения одного множества в другое.

24. Класс Vector, позволяющий хранить динамический массив целых чисел. Реализовать конструкторы, методы push_back(), insert(), erase(), операции =, + (слияние массивов), <<, >>.