Вся практика к курсу
“ООП на примере C++”

Обратите внимание: некоторые задания опираются на материалы, которые прорабатываются на семинарах и/или в методичках. Если вы выполняете задания наперед (опережая курс), то данные задания стоит отложить и дождаться доступа к соответствующим материалам.

Урок 1
“Именованные состояния. Инкапсуляция.”

1. Создать класс **Power**, который содержит два вещественных числа. Этот класс должен иметь две переменные-члена для хранения этих вещественных чисел. Еще создать два метода: один с именем **set**, который позволит присваивать значения переменным, второй — **calculate**, который будет выводить результат возведения первого числа в степень второго числа. Задать значения этих двух чисел по умолчанию.

2. Написать класс с именем RGBA, который содержит 4 переменные-члена типа **std::uint8_t: m_red**, **m_green**, **m_blue** и **m_alpha** (**#include cstdin**t для доступа к этому типу). Задать 0 в качестве значения по умолчанию для **m_red**, **m_green**, **m_blue** и 255 для **m_alpha**. Создать конструктор со списком инициализации членов, который позволит пользователю передавать значения для **m_red**, **m_blue**, **m_green** и **m_alpha**. Написать функцию **print()**, которая будет выводить значения переменных-членов.

3. Написать класс, который реализует функциональность стека. Класс **Stack** должен иметь:

    - private-массив целых чисел длиной 10;
    - private целочисленное значение для отслеживания длины стека;
    - public-метод с именем **reset()**, который будет сбрасывать длину и все значения элементов на 0;
    - public-метод с именем **push()**, который будет добавлять значение в стек. **push()** должен возвращать значение **false**, если массив уже заполнен, и **true** в противном случае;
    - public-метод с именем **pop()** для вытягивания и возврата значения из стека. Если в стеке нет значений, то должно выводиться предупреждение;
    - public-метод с именем **print()**, который будет выводить все значения стека.

Код **main()**:

```
int main()
```

```
{
    Stack stack;
    stack.reset();
    stack.print();

    stack.push(3);
    stack.push(7);
    stack.push(5);
    stack.print();

    stack.pop();
    stack.print();

    stack.pop();
    stack.pop();
    stack.print();

    return 0;
}
```

Этот код должен выводить:

( )
(3 7 5)
(3 7)
( )

Урок 2
“Наследование”

1. Создать класс **Person** (человек) с полями: имя, возраст, пол и вес. Определить методы переназначения имени, изменения возраста и веса. Создать производный класс **Student** (студент), имеющий поле года обучения. Определить методы переназначения и увеличения этого значения. Создать счетчик количества созданных студентов. В функции main() создать несколько студентов. По запросу вывести определенного человека.

2. Создать классы **Apple** (яблоко) и **Banana** (банан), которые наследуют класс **Fruit** (фрукт). У **Fruit** есть две переменные-члена: **name** (имя) и **color** (цвет). Добавить новый класс **GrannySmith**, который наследует класс **Apple**.

```
int main()
{
    Apple a("red");
    Banana b;
    GrannySmith c;

    std::cout << "My " << a.getName() << " is " << a.getColor() << ".\n";
    std::cout << "My " << b.getName() << " is " << b.getColor() << ".\n";
    std::cout << "My " << c.getName() << " is " << c.getColor() << ".\n";

    return 0;
}
```

Код, приведенный выше, должен давать следующий результат:

```
My apple is red.
My banana is yellow.
My Granny Smith apple is green.
```

3. Изучить правила игры в Blackjack. Подумать, как написать данную игру на C++, используя объектно-ориентированное программирование. Сколько будет классов в программе? Какие классы будут базовыми, а какие производными? Продумать реализацию игры с помощью классов и записать результаты.

Урок 3
“Виртуальные функции и полиморфизм”

1. Создать абстрактный класс Figure (фигура). Его наследниками являются классы Parallelogram (параллелограмм) и Circle (круг). Класс Parallelogram — базовый для классов Rectangle (прямоугольник), Square (квадрат), Rhombus (ромб). Для всех классов создать конструкторы. Для класса Figure добавить чисто виртуальную функцию area() (площадь). Во всех остальных классах переопределить эту функцию, исходя из геометрических формул нахождения площади.
2. Создать класс Car (автомобиль) с полями company (компания) и model (модель). Классы-наследники: PassengerCar (легковой автомобиль) и Bus (автобус). От этих классов наследует класс Minivan (минивэн). Создать конструкторы для каждого из классов, чтобы они выводили данные о классах. Создать объекты для каждого из классов и посмотреть, в какой последовательности выполняются конструкторы. Обратить внимание на проблему «алмаз смерти».

   Примечание: если использовать виртуальный базовый класс, то объект самого "верхнего" базового класса создает самый "дочерний" класс.

3. Создать класс: Fraction (дробь). Дробь имеет числитель и знаменатель (например, 3/7 или 9/2). Предусмотреть, чтобы знаменатель не был равен 0. Перегрузить:
   - математические бинарные операторы (+, -, *, /) для выполнения действий с дробями
   - унарный оператор (-)
   - логические операторы сравнения двух дробей (==, !=, <, >, <=, >=).

   Примечание: Поскольку операторы < и >=, > и <= — это логические противоположности, попробуйте перегрузить один через другой.

   Продемонстрировать использование перегруженных операторов.

4. Создать класс Card, описывающий карту в игре БлэкДжек. У этого класса должно быть три поля: масть, значение карты и положение карты (вверх лицом или рубашкой). Сделать поля масть и значение карты типом перечисления (enum). Положение карты - тип bool. Также в этом классе должно быть два метода:
   - метод Flip(), который переворачивает карту, т.е. если она была рубашкой вверх, то он ее переворачивает лицом вверх, и наоборот.
   - метод GetValue(), который возвращает значение карты, пока можно считать, что туз = 1.

Урок 4
“Отношения между объектами в C++”

1. Добавить в контейнерный класс, который был написан в этом уроке, методы:
    - для удаления последнего элемента массива (аналог функции pop_back() в векторах)
    - для удаления первого элемента массива (аналог pop_front() в векторах)
    - для сортировки массива
    - для вывода на экран элементов.
2. Дан вектор чисел, требуется выяснить, сколько среди них различных. Постараться использовать максимально быстрый алгоритм.
3. Реализовать класс Hand, который представляет собой коллекцию карт. В классе будет одно поле: вектор указателей карт (удобно использовать вектор, т.к. это по сути динамический массив, а тип его элементов должен быть - указатель на объекты класса Card). Также в классе Hand должно быть 3 метода:
    - метод Add, который добавляет в коллекцию карт новую карту, соответственно он принимает в качестве параметра указатель на новую карту
    - метод Clear, который очищает руку от карт
    - метод GetValue, который возвращает сумму очков карт руки (здесь предусмотреть возможность того, что туз может быть равен 11).

Урок 5

“Совместное использование функций и методов”

1. Реализовать шаблон класса **Pair1**, который позволяет пользователю передавать данные одного типа парами.

   Следующий код:

```
int main()
{
    Pair1<int> p1(6, 9);
    cout << "Pair: " << p1.first() << ' ' << p1.second() << '\n';

    const Pair1<double> p2(3.4, 7.8);
    cout << "Pair: " << p2.first() << ' ' << p2.second() << '\n';

    return 0;
}
```

   … должен производить результат:

   Pair: 6 9
   Pair: 3.4 7.8

2. Реализовать класс **Pair**, который позволяет использовать разные типы данных в передаваемых парах.

   Следующий код:

```
int main()
{
    Pair<int, double> p1(6, 7.8);
    cout << "Pair: " << p1.first() << ' ' << p1.second() << '\n';

    const Pair<double, int> p2(3.4, 5);
    cout << "Pair: " << p2.first() << ' ' << p2.second() << '\n';

    return 0;
}
```

   … должен производить следующий результат:

   Pair: 6 7.8
   Pair: 3.4 5

   Подсказка: чтобы определить шаблон с использованием двух разных типов, просто разделите параметры типа шаблона запятой.

3. Написать шаблон класса **StringValuePair**, в котором первое значение всегда типа **string**, а второе — любого типа. Этот шаблон класса должен наследовать частично специализированный класс **Pair**, в котором первый параметр — **string**, а второй — любого типа данных.

Следующий код:

```
int main()
{
    StringValuePair<int> svp("Amazing", 7);
    std::cout << "Pair: " << svp.first() << ' ' << svp.second() << '\n';
    return 0;
}
```

… должен производить следующий результат:

Pair: Amazing 7

*Подсказка: при вызове конструктора класса **Pair** из конструктора класса **StringValuePair** не забудьте указать, что параметры относятся к классу **Pair**.*

4. Согласно иерархии классов, которая представлена в методичке к уроку 3, от класса **Hand** наследует класс **GenericPlayer**, который обобщенно представляет игрока, ведь у нас будет два типа игроков - человек и компьютер. Создать класс **GenericPlayer**, в который добавить поле **name** - имя игрока. Также добавить 3 метода:
    - **IsHitting()** - чисто виртуальная функция, возвращает информацию, нужна ли игроку еще одна карта.
    - **IsBoosted()** - возвращает bool значение, есть ли у игрока перебор
    - **Bust()** - выводит на экран имя игрока и объявляет, что у него перебор.

Урок 6

“Потоки ввода-вывода”

1. Создать программу, которая считывает целое число типа **int**. И поставить «защиту от дурака»: если пользователь вводит что-то кроме одного целочисленного значения, нужно вывести сообщение об ошибке и предложить ввести число еще раз. Пример неправильных введенных строк:

   rbtrb

   nj34njkn

   1n

2. Создать собственный манипулятор **endll** для стандартного потока вывода, который выводит два перевода строки и сбрасывает буфер.

3. Реализовать класс **Player**, который наследует от класса **GenericPlayer**. У этого класса будет 4 метода:
   - **virtual bool IsHitting() const** - реализация чисто виртуальной функции базового класса. Метод спрашивает у пользователя, нужна ли ему еще одна карта и возвращает ответ пользователя в виде **true** или **false**.
   - **void Win() const** - выводит на экран имя игрока и сообщение, что он выиграл.
   - **void Lose() const** - выводит на экран имя игрока и сообщение, что он проиграл.
   - **void Push() const** - выводит на экран имя игрока и сообщение, что он сыграл вничью.

4. Реализовать класс **House**, который представляет дилера. Этот класс наследует от класса **GenericPlayer**. У него есть 2 метода:
   - **virtual bool IsHitting() const** - метод указывает, нужна ли дилеру еще одна карта. Если у дилера не больше 16 очков, то он берет еще одну карту.
   - **void FlipFirstCard()** - метод переворачивает первую карту дилера.

5. Написать перегрузку оператора вывода для класса **Card**. Если карта перевернута рубашкой вверх (мы ее не видим), вывести **XX**, если мы ее видим, вывести масть и номинал карты. Также для класса **GenericPlayer** написать перегрузку оператора вывода, который должен отображать имя игрока и его карты, а также общую сумму очков его карт.

Урок 7

“Поддержка модульности. Написание игры Blackjack.”

1. Создайте класс **Date** с полями день, месяц, год и методами доступа к этим полям. Перегрузите оператор вывода для данного класса. Создайте два "умных" указателя **today** и **date**. Первому присвойте значение сегодняшней даты. Для него вызовите по отдельности методы доступа к полям класса **Date**, а также выведите на экран данные всего объекта с помощью перегруженного оператора вывода. Затем переместите ресурс, которым владеет указатель **today** в указатель **date**. Проверьте, являются ли нулевыми указатели **today** и **date** и выведите соответствующую информацию об этом в консоль.

2. По условию предыдущей задачи создайте два умных указателя **date1** и **date2**.

- Создайте функцию, которая принимает в качестве параметра два умных указателя типа **Date** и сравнивает их между собой (сравнение происходит по датам). Функция должна вернуть более позднюю дату.
- Создайте функцию, которая обменивает ресурсами (датами) два умных указателя, переданных в функцию в качестве параметров.

Примечание: обратите внимание, что первая функция не должна уничтожать объекты, переданные ей в качестве параметров.

3. Создать класс **Deck**, который наследует от класса **Hand** и представляет собой колоду карт. Класс **Deck** имеет 4 метода:

- **vold Populate()** - Создает стандартную колоду из 52 карт, вызывается из конструктора.
- **void Shuffle()** - Метод, который тасует карты, можно использовать функцию из алгоритмов **STL random_shuffle**
- **vold Deal (Hand& aHand)** - метод, который раздает в руку одну карту
- **void AddItionalCards (GenericPlayer& aGenerIcPlayer)** - раздает игроку дополнительные карты до тех пор, пока он может и хочет их получать

Обратите внимание на применение полиморфизма. В каких методах применяется этот принцип ООП?

4. Реализовать класс **Game**, который представляет собой основной процесс игры. У этого класса будет 3 поля:

- колода карт
- рука дилера
- вектор игроков.

Конструктор класса принимает в качестве параметра вектор имен игроков и создает объекты самих игроков. В конструкторе создается колода карт и затем перемешивается.

Также класс имеет один метод **play()**. В этом методе раздаются каждому игроку по две стартовые карты, а первая карта дилера прячется. Далее выводится на экран информация о картах каждого игра, в т.ч. и для дилера. Затем раздаются игрокам дополнительные карты. Потом показывается первая карта дилера и дилер набирает карты, если ему надо. После этого выводится сообщение, кто победил, а кто проиграл. В конце руки всех игроков очищаются.

5. Написать функцию **main()** к игре **Блэкджек**. В этой функции вводятся имена игроков. Создается объект класса **Game** и запускается игровой процесс. Предусмотреть возможность повторной игры.

Урок 8

“Механизм исключительных ситуаций”

1. Написать шаблонную функцию **div**, которая должна вычислять результат деления двух параметров и запускать исключение **DivisionByZero**, если второй параметр равен 0. В функции **main** выводить результат вызова функции **div** в консоль, а также ловить исключения.

2. Написать класс **Ex**, хранящий вещественное число **x** и имеющий конструктор по вещественному числу, инициализирующий **x** значением параметра. Написать класс **Bar**, хранящий вещественное число **y** (конструктор по умолчанию инициализирует его нулем) и имеющий метод **set** с единственным вещественным параметром **a**. Если **y + a > 100**, возбуждается исключение типа Ex с данными **a*y**, иначе в **y** заносится значение **a**. В функции **main** завести переменную класса **Bar** и в цикле в блоке **try** вводить с клавиатуры целое **n**. Использовать его в качестве параметра метода **set** до тех пор, пока не будет введено 0. В обработчике исключения выводить сообщение об ошибке, содержащее данные объекта исключения.

3. Написать класс «робот», моделирующий перемещения робота по сетке 10x10, у которого есть метод, означающий задание переместиться на соседнюю позицию. Эти методы должны запускать классы-исключения **OffTheField**, если робот должен уйти с сетки, и **IllegalCommand**, если подана неверная команда (направление не находится в нужном диапазоне). Объект исключения должен содержать всю необходимую информацию — текущую позицию и направление движения. Написать функцию **main**, пользующуюся этим классом и перехватывающую все исключения от его методов, а также выводящую подробную информацию о всех возникающих ошибках.