**Лабораторная работа №15**

**Выбор и обоснование выбора среды разработки программы. Изучение различных стилей программирования, правил формирования листинга программы.**

**Задание1.**

Наш программный продукт будет разработан на таких языках программирования как C#, HTML, CSS, JS и MySQL. Данные языки программирования были выбраны по следующим критериям:

1. Скорость работы конечного продукта;
2. Объем занимаемой оперативной памяти;
3. Скорость разработки программы;
4. Ориентированность на компьютер или человека;
5. Кроссплатформенность;
6. Скорость внесения изменений, скорость тестирования.

**C#** – это объектно-ориентированный, простой и в то же время мощный язык программирования, который позволяет разработчикам создавать многофункциональные приложения.

Достоинства HTML:

* сохраняет информацию о выбранных автором шрифтах, кодировках, стилях, цвете текста;
* читается на различных компьютерных платформах;
* многие программы (в их числе бесплатные) умеют с ним работать;
* это открытый стандарт.

Преимущества использования CSS

* Универсальность;
* Альтернативность вставки;
* Ускорение;
* Кэширование;
* Преемственность;
* Помощь и поддержка.

**Javascript**популярен не случайно, а благодаря своим несомненным достоинствам:

* Незаменимость для веб-разработки. Поддержка скриптов всеми популярными браузерами; полная интеграция с вёрсткой страниц (HTML+CSS) и серверной частью (backend).
* Скорость работы и производительность. Javascript позволяет частично обрабатывать веб-страницы на компьютерах пользователя без запросов к серверу. Это экономит время и трафик, снижает нагрузку на сервер.

Преимущества MySQL:

* Легко использовать
* Предоставляет большой функционал
* Хорошие функции безопасности
* Легко масштабируется и подходит для больших баз данных
* Обеспечивает хорошую скорость и производительность
* Обеспечивает хорошее управление пользователями и множественный контроль доступа

**Задание2.**

При создании программного продукта буду использованные такие парадигмы как модульное и объектно-ориентированное программирование.

Преимущества модульного программирования

* Избегайте глобального загрязнения переменных и конфликтов имен
* Повысьте частоту повторного использования кода
* Повышение удобства сопровождения кода
* Совместная работа нескольких человек не мешает друг другу

Преимущества объектно-ориентированного программирования:

* Легко читается. Не нужно выискивать в коде функции и выяснять, за что они отвечают;
* Быстро пишется. Можно быстро создать сущности, с которыми должна работать программа;
* Проще реализовать большой функционал. Так как на написание кода уходит меньше времени, можно гораздо быстрее создать приложение с множеством возможностей;
* Меньше повторений. Не нужно писать однотипные функции для разных сущностей.

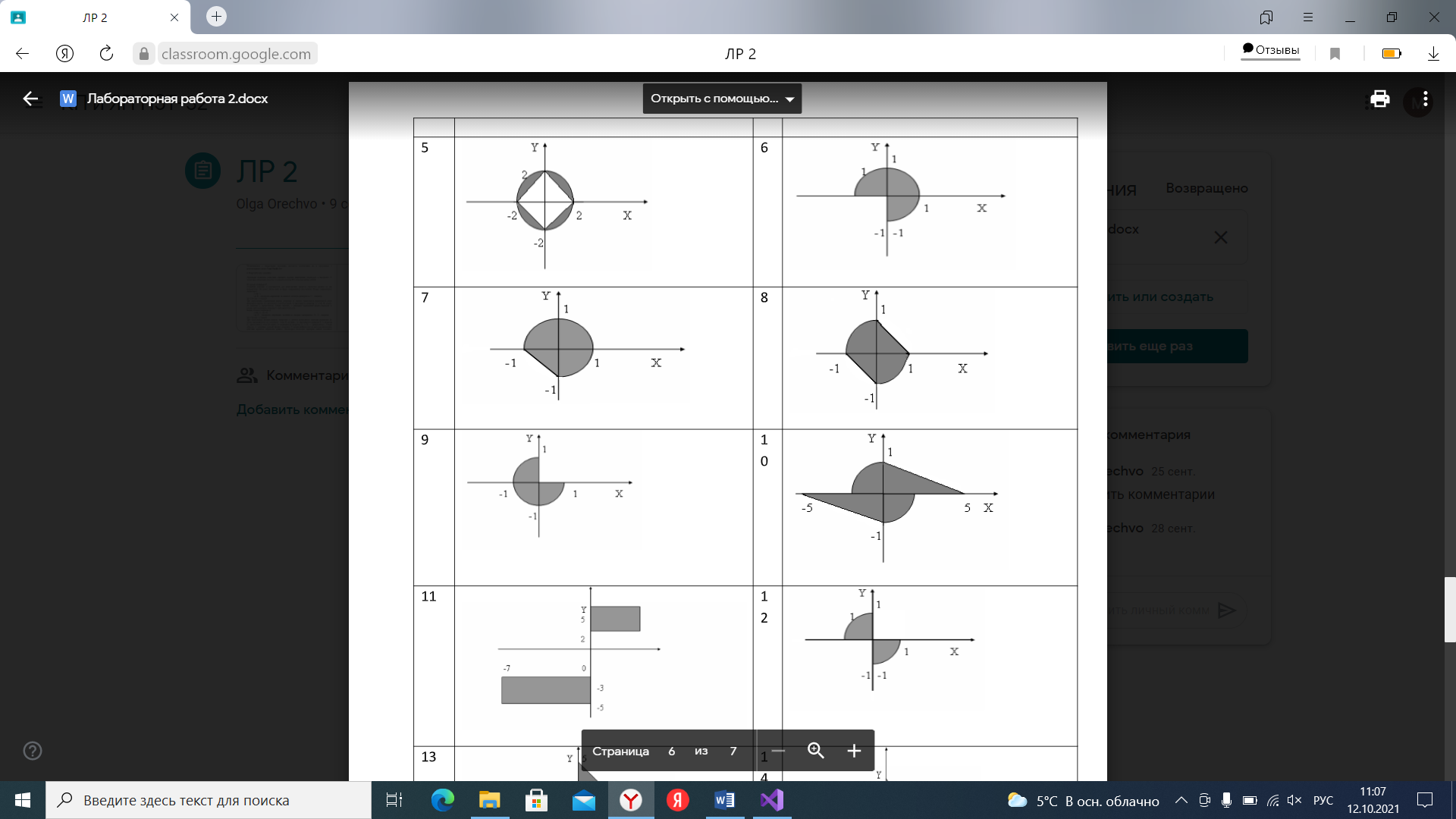
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название парадигмы | Сущность парадигмы, основые идеи | Языки, поддерживающие даную парадигму | Достоинства | Недостатки |
| Императивное пограмирование | Используются утверждения, инструкции, последова-тельное изменение состояния | Pascal, C, Basic | Функцио-  Нальное, ло-  гическое | Невозможность исп. Код повторно |
| Декларативное программирование | Парадигма программирования, в которой задается спецификации решения задачи, т.е. описывается, что представляют собой проблема и ожидаемый результат. | HTML, SQL | Они позволяют вам работать на гораздо более высоком уровне абстракции; | Декларативные языки программирования могут легко создать узкие места в производительности; отсутствует возможность адекватно выразить в программе естественный параллелизм задачи. |
| Структурное программирование | Парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. | Java, Python, JS, Ruby | Повышенная надежность программ; повышается эффективность программ; уменьшается время и стоимость программной разработки; улучшается читабельность программ | Локальные модификации могли нарушить работоспособность всей системы; невозможно полностью отказаться от меток и операторов безусловного перехода при создании сложных ПП; процессы и данные существуют отдельно друг от друга, причем проектирование ведется от процессов к данным. |
| Процедурное программирование | программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка. | C, C++, C#, Pascal, Basic, Java, Fortran, JS | Задает общую структуру программы: 1) данные 2) и процедуры; процедуры можно использовать повторно; можно даже создавать библиотеки многократно используемых процедур. | Слабо поддерживают многократное использование; процедуры очень зависят от обрабатываемых ими данных. |
| Модульное программирование | метод проектирования программного обеспечения, который подчеркивает разделение функциональности программы на независимые взаимозаменяемые модули, так что каждый из них содержит все необходимое для выполнения только одного аспекта желаемой функциональности. | Python, Pascal, C++, Java, C# | Можно создавать библиотеки наиболее употребительных модулей; появляется много естественных контрольных точек для отладки проекта; Можно создавать библиотеки наиболее употребительных модулей; появляется много естественных контрольных точек для отладки проекта; | Возрастает размер требуемой оперативной памяти; увеличивается время компиляции и загрузки; довольно сложными становятся межмодульные интерфейсы; увеличивается время выполнения программы. |
| Объектно- ориентированное программирование | Методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. | Ada, C++, Java,Python | Сокращается время на разработку, которое может быть отдано другим задачам; компоненты многоразового использования обычно содержат гораздо меньше ошибок, чем вновь разработанные, ведь они уже не раз подвергались проверке. | Некоторое снижение производительности функционирования ПО и высокие начальные затраты |
| Функциональное программирование | Парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних | Haskell, LISP, Scala, R, Python | Функциональное программирование имеет надежное математическое обоснование; возможность оптимизации и верификации программ; простота механизма перехода от простых объектов к более сложным; возможность стандартизации  параллельной обработки информации. | Неформальный подход при разработке программ; неудобства при работе с БД. |
| Логическое пограммирование | Парадигма программирования, основанная на автоматическом доказательстве теорем, а также раздел дискретной математики | Prolog, Planner, Conniver | Операции, совершаемые в логическом программировании всегда понятны  Результат практически всегда не зависит от выбранного пути реализации  Может быть использован в качестве невычислительного языка используя только выражения и факты | 1. Если брать за пример логического языка программирования Prolog, то на лицо невозможность создания комплексных задач. То есть в реальности логический язык может идти дополнением к процедурному, но самостоятельно используется крайне редко; 2. Из-за недостатка в инвестициях и простом внимании, логические языки слабо развиваются; 3. Если предстоит иметь дело с вычислительными операциями, то логические языки программирования - не лучший выбор. |

**Задание3.**

**Структурное программирование:**

**Условие:** Дана точка на плоскости с координатами (х, у). Составить программу, которая выдает одно

из сообщений «Да», «Нет», «На границе» в зависимости от того, лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе. Области задаются графически следующим образом:



using System;

namespace лаб.раб\_2.\_1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("x= ");//вывод на экран сообщения пользователю

double x = double.Parse(Console.ReadLine());//ввод переменной

if (x==-2)//условие

{

Console.WriteLine("Функция не определена");//если усл. верно. выводим сообщение

}

else

{

double y = 3 / Math.Abs(Math.Pow(x, 3) + 8);//если усл. неверно, считаем след. условие

Console.WriteLine("Y=");//выводим сообщение

Console.Write(y);//выводим переменную

}

}

}

}

**Модульное программирование:**

Условие: проверить, являются ли все части числа простыми

void super(int n){

 long v,a; int i;

 for (v=1,i=0; i<n; i++) v\*=10;              // Определение нижней границы

 for (a=v/10; a<v; a++){

if (PR(a)==0) continue;

            for (long ll=10; ll<v; ll\*=10){     // ll пробегает значения 10,100, 1000 < v

if (PR(a/ll)==0)          // Проверка старшей части

                        break;                      // Не простое досрочный выход

                        if (PR(a%ll)==0)        // Проверка младшей части

                        break;                      // Не простое досрочный выход

                        }

            if (ll==v)                                     // Достигли конца все простые

printf("super=%ld\n",a);

}}

**Объектно-ориентированное программирование:**

Пример ООП:

**class** DemoClass

{

// Поле класса

**int** field = 0;

// Свойство класса

**public** **int** Property {**get**;**set**;}

// Метод класса

**public** **void** Method()

{

Console.WriteLine("Method");

}

}