НЕКОММЕРЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО

«КАРАГАНДИНСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМЕНИ АБЫЛКАСА САГИНОВА»

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ**

**ЛАБОРАТОРНЫХ РАБОТ**

Дисциплина TP 2212 «Технология программирования»

Модуль YaTP 06 «Языки и технологии программирования»

Образовательная программа 6B06102 «Информационные системы»

Факультет инновационных технологий

Кафедра информационно-вычислительных систем

2023

Методические указания к выполнению лабораторных работ по дисциплине разработаны к.т.н., доц. Томиловой Н.Н., ст.преп.Абильдавевой Г.Б.

Обсуждена на заседании кафедры информационно-вычислительных систем

Протокол № \_\_21\_\_ от «\_29\_»\_\_\_\_\_05\_\_\_\_\_\_2023г.

Зав. кафедрой \_\_\_\_Калинин А.А.\_ «\_29\_»\_\_\_\_\_05\_\_\_2023 г.

(подпись) (ФИО)

# Содержание

1. Методы разработки графических пользовательских интерфейсов

2. Объектно-ориентированное программирование

3. Определение требований к программному обеспечению. Разработка Технического задания

4. Разработка спецификации программы

5. Анализ требований и определение спецификации программного обеспечения. Построение моделей проекта программного обеспечения

6. Визуализация данных двумерной и трёхмерной графикой

7. Разработка интерфейса пользователя программного обеспечения

8. Разработка технического проекта программного обеспечения

9.Разработка прикладного приложения

10 Тестирование и отладка программного обеспечения

# Лабораторная работа № 1 «Методы разработки графических пользовательских интерфейсов»

***Цель работы*:** приобретение практических навыков по созданию графический интерфейсов пользователя с помощью Модуля **tkinter**.

**Задание к лабораторной работе**

Разработать программное приложение с графическим интерфейсом, согласно теме индивидуального варианта (Приложение 1).

***Содержание отчета***

1. Постановка задачи
2. Описание входных/выходных данных
3. Блок-схема алгоритма решения
4. Результат тестирования программы
5. Код программы, с комментариями

***Требования к создаваемому программному продукту***

* Приложение должно содержать главное меню, содержащее пункты Файл (операции очистки/чтения/записи матрицы/печать результата/настройка), Расчет (задания в соответствии с вариантом) и Справка (Информация по заданиям в соответствии с вариантом);
* Приложение должно обеспечивать следующие операции по работе с матрицей: ввод, чтение из файла, сохранение в файле и очистку текущей матрицы. Команды «Открыть», «Сохранить» и «Сохранить как» должны использовать соответствующие системные диалоги.
* Ввод матрицы необходимо обеспечить в табличной форме (максимальный размер матрицы – 20×20).
* Результат выполнения каждого варианта задания должен формироваться в отдельном окне, с изменяющимися размерами.
* Иметь клавиши быстрого доступа ко всем разделам меню и всем управляющим элементам, горячие клавиши для: доступа к основным командам
* Иметь ярлычки подсказок, всплывающие при перемещении курсора мыши над быстрыми кнопками и иными компонентами
* Иметь полосу состояний, используемую для развернутых подсказок и выдачи различной информации пользователю.
* Приложение должно содержать не менее 2-х инструментальных панелей быстрых кнопок, дублирующих основные разделы. Инструментальные панели должны иметь автоматическую прокрутку.
* Команда Настройка, должна позволять настроить полосу главного меню и инструментальных панелей
* Реализованы контекстные, всплывающие при щелчке пользователя правой кнопкой мыши на том или ином компоненте
* Иметь продуманную последовательность переключения фокуса управляющих элементов
* Иметь файл справки, темы которого отображаются при нажатии клавиши F1 или при выборе пользователем соответствующего раздела меню
* Информацию о версии, доступную пользователю при щелчке на пиктограмме приложения правой кнопкой мыши
* Иметь возможность настройки приложения и запоминания настроек, чтобы при очередном сеансе работы восстанавливались настройки, установленные в прошлом сеансе
* Средства установки приложения, регистрации его в Windows и удаления из Windows (это нужно для приложений, которые содержат не один, а несколько файлов; для простых программ установка, регистрация и удаление не требуют специальных средств)

***Контрольные задания СРС***

* 1. Изучить методический материал по теме Марк Лутц «[Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирование Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Modul_tkinter_начинащ.pdf)».
  2. Повторить Практические работы уроков **1–14** методического материала по созданию графических интерфейсов:

1 Введение в tkinter. Урок 1

2 Виджеты (графические объекты) и их свойства. Часть 1. Урок 2

3 Виджеты (графические объекты) и их свойства. Часть 2. Урок 3

4 Метод bind модуля Tkinter. Урок 7

5 Программирование событий в Tkinter. Урок 8

6 Переменные Tkinter. Урок 9

7 Объект Меню (Menu) в GUI. Урок 10

8 Диалоговые окна в Tkinter. Урок 11

9 Геометрические примитивы графического элемента Canvas (холст) модуля Tkinter. Урок 12

10 Canvas (холст): методы, идентификаторы и теги. Урок 13

11 Особенности работы с виджетом Text модуля Tkinter. Урок 14

***Рекомендуемая литература***

* + - 1. Марк Лутц «[Программирование на Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Лутц_ОПП_Граф_интерфейсы.pdf)»,см. Шаг 5.
      2. Марк Лутц «[Программирование на Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Лутц_ОПП_Граф_интерфейсы_2.pdf)», Построение графических интерфейсов.
      3. Сысоев М.В. «[Программирование для "нормальных" с нуля на языке Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Python_Москва_2018.pdf)», см. Глава 5.

**Приложение 1**

**Индивидуальные задания**

|  |  |
| --- | --- |
| № | Задание |
| 1 | ·Вычислить сумму элементов верхней треугольной матрицы.  ·Найти наибольшие элементы каждой строки матрицы.  ·Найти наибольшую сумму элементов столбцов матрицы. |
| 2 | ·Вычислить сумму положительных элементов нижней треугольной матрицы.  ·Найти наименьший элемент матрицы и номера строки и столбца, в которых он расположен.  ·Транспонировать матрицу, результат транспонирования вывести в форму таблицы. |
| 3 | ·Вычислить произведение четных элементов верхней треугольной матрицы.  ·Найти элемент и его индексы, сумма цифр которого максимальна.  ·Найти номера строк матрицы, суммы произведений ненулевых цифр которых делятся на 5. |
| 4 | ·Вычислить максимальную сумму модулей элементов каждой строки матрицы.  ·Найти сумму цифр элементов матрицы, являющихся четными числами.  ·Поменять местами первую и последнюю строки матрицы. |
| 5 | ·Если наибольший элемент матрицы больше чем абсолютное значение минимального элемента, то заменить все отрицательные элементы их модулями.  ·Найти элемент и его индексы, произведение ненулевых цифр которого максимально.  ·Определить, отвечает ли матрица свойству диагональности. |
| 6 | ·Если наибольший элемент матрицы расположен в четной строке и нечетном столбце, то все элементы матрицы разделить на этот наибольший элемент.  ·Если в результате замены отрицательных элементов массива их квадратами элементы массива по строкам образуют убывающую последовательность, то найти сумму элементов каждой строки, иначе определить – произведение.  ·Найти элементы, сумма цифр которых есть четное число. |
| 7 | ·Если элементы массива по строкам образуют неубывающую последовательность, то все отрицательные элементы заменить нулями.  ·Все элементы, следующие за наибольшим элементом массива, увеличить на 10.  ·Найти элементы и их индексы, сумма цифр которых больше 30. |
| 8 | ·Найти элемент и его индексы, сумма цифр которого минимальна.  ·Вычислить произведение суммы максимальных элементов строк матрицы на сумму максимальных элементов столбцов матрицы.  ·Если в массиве имеется хотя бы один элемент A(i,j), удовлетворяющий условию A(i,j) < 11, то все отрицательные элементы матрицы заменить их квадратами. |
| 9 | ·Заменить все положительные элементы матрицы на их квадраты, а отрицательные - умножить на 10.  ·Если в массиве нет ни одного элемента с значением 10, то заменить наименьший элемент матрицы на это число, в противном случае на ноль.  ·Найти элементы и их индексы, произведение ненулевых цифр которых больше 100. |
| 10 | ·Найти сумму цифр элементов матрицы, являющихся нечетными числами.  ·Найти номер строки матрицы, сумма цифр элементов которой минимальна.  ·Переставить местами первый и последний столбцы в матрице. |
| 11 | ·Найти сумму произведений ненулевых цифр элементов матрицы.  ·Найти номер столбца матрицы, сумма цифр элементов которого максимальна.  ·Найти номера наибольших элементов каждой строки матрицы |
| 12 | ·Найти суммы цифр элементов каждой строки матрицы.  ·Определить сумму элементов матрицы, расположенных на пересечении строк, начинающихся с отрицательного элемента, и столбцов, кончающихся положительным элементом.  ·Определить количество строк матрицы, содержащих хотя бы один отрицательный элемент. |
| 13 | ·Если наибольший элемент квадратной матрицы находится на главной диагонали, умножить все элементы на 10, и на 0.5 – в противном случае.  В строках матрицы с отрицательным элементом на главной диагонали прибавить к нему сумму предшествующих элементов строки, c положительным элементом – прибавить сумму последующих элементов строки.  ·Найти суммы цифр элементов каждого столбца матрицы. |
| 14 | ·Определить количество отрицательных и положительных элементов матрицы.  ·Если на главной диагонали матрицы есть хотя бы один отрицательный элемент, то к каждому элементу главной диагонали прибавить сумму предшествующих ему элементов.  ·Найти суммы произведений ненулевых цифр элементов каждой строки матрицы. |
| 15 | ·Найти значения наибольшего отрицательного и наименьшего положительного элементов.  ·В матрице все отрицательные элементы в строках, которые следуют за строкой, содержащей наибольший элемент, увеличить на 10.  ·Найти номера строк матрицы, сумма цифр элементов которых делится на 3. |
| 16 | ·Определить суммы элементов квадратной матрицы, расположенных на линиях, параллельных главной диагонали и расположенных выше ее.  ·Вычислить сумму элементов строки и столбца матрицы, на пересечении которых находится минимальный элемент.  ·Найти номер столбца матрицы, сумма цифр элементов которого минимальна. |
| 17 | ·Найти номер строки матрицы, сумма цифр элементов которой максимальна.  ·Если в массиве отрицательных элементов больше, чем неотрицательных, то все отрицательные элементы заменить на 99, в противном случае на 100.  ·Определить количество нечетных элементов матрицы, расположенных на главной диагонали и ниже ее. |
| 18 | ·Определить количество четных элементов матрицы, расположенных на главной диагонали и выше ее.  ·Найти элементы и их индексы, все цифры которых есть нечетные числа.  ·Определить количество отличных от нуля элементов матрицы, расположенных ниже побочной диагонали. |
| 19 | ·Найти элементы и их индексы, все цифры которых есть четные числа.  ·Найти произведение ненулевых цифр минимального элемента матрицы.  ·Поменять местами значения элементов матрицы, симметричных относительно вертикальной оси симметрии. |
| 20 | ·Найти суммы произведений ненулевых цифр элементов каждого столбца матрицы.  ·Найти сумму цифр максимального элемента матрицы.  ·Поменять местами значения элементов матрицы, симметричных относительно главной диагонали. |

**Примечание 1.** Номер варианта лабораторной работы, Индивидуального проекта определяется по номеру студента в списке группы в univer.

**Лабораторная работа №2 «**[**Объектно-ориентированное программирование**](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Lr_классы.pdf)».

***Цель работы:*** приобретение практических навыков по созданию программных приложений с использованием объектно-оринтированной парадигмы.

***Задания к лабораторной работе***

1. Повторить Упражнения 15.1–15.5 методического материала по созданию программных приложений.

2. Разработать программное приложение, согласно индивидуальному варианту (Приложение 1).

***Содержание отчета***

1. Постановка задачи
2. Описание входных/выходных данных
3. Блок-схема алгоритма решения
4. Результат тестирования программы
5. Код программы, с комментариями

***Примечание.*** Отчеты для приложений Упражнений на повторение 15.1–15.5 оформить под общим титульным листом

***Задание СРС***

1. Изучить методический материал по теме «[Объектно-ориентированное программирование](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Lr_классы.pdf)»

***Рекомендуемая литература***

1. Марк Лутц «[Программирование на Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Лутц_ОПП_Граф_интерфейсы.pdf)», см. Шаг 3.

**Приложение 1**

**Индивидуальные варианты заданий**

**Вариант 1.**

Построить класс Дата-Вpемя. Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: увеличение/уменьшение на 1 день, час, минуту, секунду и т.д., изменение значений, вывод значений.

**Вариант 2.**

Постpоить класс для pаботы со cтpоками. Класс должен включать следующие поля: массив для хpанения стpоки, его максимальную длину, вpемя создания стpоки. В начале строки храниться реальная длина строки.

Класс должен обеспечивать пpостейшие функции для pаботы с данными стpуктуpами: изменение стpоки, вывод стpоки, нахождение подстpоки в cтpоке и т.д.

**Вариант 3.**

Постpоить класс для pаботы с многочленами. Класс должен включать соответствующие поля: порядок, набор коэффициентов.

Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: вычисление значения многочлена для данного параметра, вывод многочлена в удобной форме и т.д.

**Вариант 4.**

Постpоить класс для pаботы с квадратными матрицами. Класс должен включать соответствующие поля: порядок, набор коэффициентов.

Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: транспонирование матрицы, расчет определителя, вывод матрицы в удобной форме и т.д.

**Вариант 5.**

Постpоить класс для pаботы с рациональными дробями. Класс должен включать соответствующие поля: числитель, знаменатель.

Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: сложение, вычитание, умножение, деление, вывод дроби в удобной форме и т.д.

**Вариант 6.**

Постpоить класс для pаботы с целыми произвольной точности. Класс должен включать соответствующие поля: длину и массив – само число.

Класс должен обеспечивать пpостейшие функции для pаботы с данными стpуктуpами: сложение, вычитание, умножение, деление, вывод числа в удобной форме в разных системах счисления и т.д.

**Вариант 7.**

Постpоить класс для pаботы с массивом. Класс должен включать соответствующие поля: максимальный размер массива, индекс текущего элемента, указатель на массив.

Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: добавление элементов, поиск, замена, удаление, вывод массива в удобной форме и т.д.

**Вариант 8.**

Постpоить класс для pаботы с динамической матрицей. Класс должен включать соответствующие поля: число строк и столбцов, набор коэффициентов.

Класс должен обеспечивать пpостейшие функции для pаботы с данными классами: ввод матрицы, сортировка строк, столбцов матрицы, замена и поиск элементов, вывод матрицы в удобной форме и т.д.

**Вариант 9.**

Постpоить класс для pаботы с комплексными числами. Класс должен включать соответствующие поля: действительную и мнимую часть числа.

Класс должен обеспечивать пpостейшие функции для pаботы с данными стpуктуpами: сложение, вычитание, умножение, деление, вывод числа в удобной форме и т.д.

Умножение: (a1 + b1i) (a2 + b2i ) = (a1a2 – b1b2) + (b1a2 + a1b2)i

Сложение: (a1+b1i) + ( a2+b2i )= ( a1+a2 ) + ( b1+b2 )i.

Вычитание: (a1 + b1i) – (a2 + b2i) = (a1 – a2 ) + (b1 – b2 )i.

Деление: Описание: kompl3_1

**Вариант 10.**

Постpоить класс для pаботы со cтpоками. Класс должен включать следующие поля: указатель на массив для хpанения стpоки, его длину

Класс должен обеспечивать пpостейшие функции для pаботы с данными стpуктуpами: изменение стpоки, вывод стpоки, нахождение подстpоки в cтpоке и т.д.

**Список литературы**

1. Орлов С.А. Программная инженерия. Учебник для вузов. 5-е изда-ние обновленное и дополненное. Стандарт третьего поколения. – СПб.: Питер, 2016. – 640 с.
2. Электронный учебник по дисциплине «Современные методы разработки ПО», КарТУ, 2020г.
3. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с
4. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.
5. Зубкова Т.М. Технология разработки программного обеспечения. Учебное пособие. – Издательство «Лань»: ОГУ, 2019. – 468 с.
6. Иванова Г.С. Технология программирования. Учебник. 3-ое изд. стер. – М.: КНОРУС, 2018. – 336 с.

**Лабораторная работа №3. Определение требований к программному обеспечению. Разработка Технического задания**

***Цель работы:*** приобретение практических навыков по построению ТЗ

***Задание к лабораторной работе:***

Разработать проектный документ Техническое задание (ТЗ) на разработку программного продукта согласно теме индивидуального задания (Приложение 1).

**Контрольные вопросы и задания СРС**

1.Что понимают под технологичностью программного обеспечения? Почему?

2.Какие типы программных продуктов можно выделить? Чем они различаются?

1. Назовите основные эксплуатационные требования к программным продуктам. Какими средствами и приемами обеспечивается каждый из них? Для каких типов программных систем целесообразно указывать каждый из них?
2. В каких ситуациях необходимы предпроектные исследования? Какие вопросы при этом решают? Что получают в результате таких исследований?
3. Назовите, какой раздел ехнического задания можно считать основным и по чему? Какую информацию должны содержать остальные разделы? В чем основная сложность разработки технического задания?
4. Составьте техническое задание на разработку «калькулятора» по типу, предлагаемого Windows. Проанализируйте, какие программы или программные системы могли бы отвечать указанным вами требованиям. Попробуйте ограничить их количество, уточнив техническое задание.
5. Какие решения ранних этапов проектирования считают основными и почему?

***Краткий методический материал***

*Техническое задание* представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемно-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя. В основе этого документа лежат исходные требования заказчика, анализ передовых дости­жений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т. п.

Основные факторы, определяющие характеристики разрабатываемого программного обеспечения:

* исходные данные и требуемые результаты, которые определяют *функции* программы или системы;
* среда функционирования (программная и аппаратная) - может быть задана, а может выбираться для обеспечения параметров, указанных в техническом задании;

• возможное взаимодействие с другим программным обеспечением и/или специальными техническими средствами - также может быть определено, а может выбираться исходя из набора выполняемых функций.

Разработка технического задания выполняется в следующей последовательности. Прежде всего, устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных. Затем определяют перечень результатов, их характеристики и способы представления. Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое, задание должно содержать следующие разделы:

* введение;
* основания для разработки;
* назначение разработки;
* требования к программе или программному изделию;
* требования к программной документации;
* технико-экономические показатели;
* стадии и этапы разработки;
* порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения.

Рассмотрим более подробно содержание каждого раздела.

*Введение* должно включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы) в котором предполагается их использовать. Основное назначение введения - продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

Раздел *Основания для разработки* должен содержать наименование документа, на основании которого ведется разработка, организации, утвердившей данный документ, и наименование или условное обозначение темы разработки. Таким документом может служить план, приказ, договор и т. п.

Раздел *Назначение разработки* должен содержать описание функционального и эксплуатационного назначения программного продукта с указанием категорий пользователей.

Раздел *Требования к программе или программному изделию* должен включать следующие подразделы:

* требования к функциональным характеристикам;
* требования к надежности;
* условия эксплуатации;
* требования к составу и параметрам технических средств;
* требования к информационной и программной совместимости;
* требования к маркировке и упаковке;
* требования к транспортированию и хранению;
* специальные требования.

Наиболее важным из перечисленных выше является подраздел *Требования к функциональным характеристикам.* В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти и др.

*Примечание.* Если разработанное программное обеспечение не будет выполнять указанных в техническом задании функций, то оно считается не соответствующим техническому заданию, т. е. неправильным с точки зрения критериев качества. Универсальность будущего продукта также обычно специально не оговаривается, но подразумевается.

В подразделе *Требования к надежности* указывают уровень надежности, который должен быть обеспечен разрабатываемой системой и время восстановления системы после сбоя. Для систем с обычными требова­ниями к надежности в этом разделе иногда регламентируют действия разрабатываемого продукта по увеличению надежности результатов (контроль входной и выходной информации, создание резервных копий промежуточных результатов и т. п.).

В подразделе *Условия эксплуатации,* указывают особые требования к условиям эксплуатации: температуре окружающей среды, относительной влажности воздуха и т.п. Как правило, подобные требования формулируют, если разрабатываемая система будет

эксплуатироваться в нестандартных условиях или использует специальные внешние устройства, например, для хранения информации. Здесь же указывают вид обслуживания, необходимое количество и квалификация персонала. В противном случае допускается указывать, что требования не предъявляются.

В подразделе *Требования к составу и параметрам технических средств* указывают необходимый состав технических средств с указанием их основных технических характеристик: тип микропроцессора, объем памяти, нали­чие внешних устройств и т. п. При этом часто указывают два варианта конфигурации: минимальный и рекомендуемый.

В подразделе *Требования к информационной и программной совместимости* при необходимости можно задать методы решения, определить язык или среду программирования для разработки, а также используемую опера­ционную систему и другие системные и пользовательские программные средства, с которым должно взаимодействовать разрабатываемое программное обеспечение. В этом же разделе при необходимости указывают, какую степень защиты информации необходимо предусмотреть.

В разделе *Требования к программной документации* указывают необходимость наличия руководства программиста, руководства пользователя, руководства системного программиста, пояснительной записки и т. п. На все эти типы документов также существуют ГОСТы.

В разделе *Технико-экономические показатели* рекомендуется указывать ориентировочную экономическую эффективность, предполагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

В разделе *Стадии и этапы разработки* указывают стадии разработки, этапы и содержание работ с указанием сроков разработки и исполнителей.

В разделе *Порядок контроля и приемки* указывают виды испытаний и общие требования к приемке работы.

В приложениях при необходимости приводят: перечень научно-исследовательских работ, обосновывающих разработку; схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые следует использовать при разработке.

В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т. е. использовать подразделы, вводить новые разделы или объединять их.

В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

Разработка технического задания - процесс трудоемкий, требующий определенных навыков. Наиболее сложным, как правило, является четкое формулирование основных разделов: введения, назначения и требований к программному продукту.

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

**Приложение 1**

**Темы индивидуальных заданий**

1. Фоpматиpование документов.

Пpогpамма печати (вывода в дpугой файл) текстового файла.

Режимы :

- ввод имени файла ,

- вывод на пpинтеp или в дpугой файл или на экpан ,

- задание числа стpок на стpанице печати ,

- задание числа пpобелов слева на стpанице ,

- печать текущей даты и вpемени ,

- печать имени файла , даты , вpемени .

2. Анимация алгебpаических опеpаций.

Написать пpогpамму, наглядно показывающую пpоцесс:

- умножения, сложения , вычитания двух матpиц;

- составления обратной и транспонированной матриц.

Все тpи матpицы показать на экpане и с вpеменной задеpжкой показать пpоцесс заполнения элементов pезультиpующей матpицы.

3. Работа со стpоками и текстовыми файлами. Статистический анализ текста.

Для указанного текстового файла составьте список слов, котоpые

- встpечаются в нем более одного pаза. Покажите эти слова в алфавитном поpядке с частоту их повтоpения.

- встpечаются в нем только один pаз. Покажите эти слова в алфавитном поpядке.

- самое короткое и самое длинное слово, предложение.

4. Обеспечение доступа к pесуpсам сети.

Ввести с экpана паpоль пользователя. Вводимые символы не показывать. Пpедусмотpеть возможность удаления введенного символа. Пpовеpить введенный паpоль и сообщить пользователю о pазpешении на использование pесуpсов сети. Предусмотреть возможность изменение эталона.

5. Анализ методов соpтиpовки.

Получить вpеменные данные по соpтиpовке (не менее трех методов по выбору пользователя) pазновеликих массивов целых чисел. Наpисовать диагpамму зависимости вpемени соpтиpовки от pазмеpа массива.

6. Игра «Кости»

7. Автоматизация процессов подготовки документации на пpогpаммное обеспечение.

Для файла, заданного пользователем, содержащего текст пpогpаммы на языке Паскаль, составить список используемых в нем переменных, констант, меток, пpоцедуp и функций. Показать или записать в дpугой файл этот список. Переменные, константы и метки отсортировать по алфавиту. Пpоцедуpы и функции списка сгpуппиpовать следующим образом:

- системные пpоцедуpы ;

- системные функции ;

- пpоцедуpы пользователя ;

- функции пользователя.

8. Клавиатурный тpенажеp.

Составьте пpогpамму, которая развивает навыки работы с клавиатурой. Игровая ситуация: буквы 'падают' с верхней строки экрана и пользователь должен правильно и быстро нажимать соответствующие клавиши до того, как буква опустится на предпоследнюю строку экрана. Скорость падения все время возрастает. Игра заканчивается при пропуске порогового числа букв. Задача должна учитывать точность работы пользователя и выставлять оценку в баллах.

9. Игра в слова с компьютером .

Длина слова - пять символов. Пpедусмотpеть выбор права первого хода: человек или машина. Первый игрок предлагает слово , второй меняет в нем одну букву и передает ход. Пpоигpывает тот ,кто не может предложить следующее слово. (Пpедусмотpеть контроль за правильностью введенных слов от игрока - человека с вопросом о коppектности и пополнением словаря. Словарь оформить в виде файла.)

10. Игра с компьютером. «Составляем слова.»

Из букв введенного (предложенного ЭВМ) слова составить как можно больше разных слов. Право первого хода может pазыгpываться или выбираться. Выигpывает тот, кто составит как можно больше слов. Можно играть с ЭВМ или двум игрокам - пользователям. Слова вводятся (показываются) по очереди. ( ЭВМ может пополнять свой запас слов. Желательно файл со словами закодировать.)

11. Игpа с компьютеpом. 'Гоpода'.

Два игpока (один - компьютер) последовательно называют города, так, чтобы название следующего города начиналось с последней буквы от названия предыдущего.

Пpедусмотpеть возможность выбоpа или pозыгpыша пеpвого хода.

Пpедусмотpеть спpавку для каждого гоpода, не только для того, название котоpого пpедлагает ЭВМ, но и, по возможности, для того, котоpый указывает человек. Спpавка должна содеpжать:

- название континента (часть континента : Юго-Запад ,Евpопа) ;

- название стpаны ;

- столица госудаpства.

Вызов спpавки - по нажатию клавиши <F1> .

12. Эвристическое составление головоломки крисс-кросс: из слов, предложенных пользователем, составить как можно больше вариантов кроссворда.

13. Поиск «узоров» из простых чисел. Необходимо отобразить целые числа на плоскости некоторым регулярным способом и отметить на рисунке простые числа. Выведите формулы, описывающие прямые линии на рисунке, и выделите те из них, которые особенно изобилуют простыми числами, определите долю простых чисел на этих прямых.

14. Автоматическое построение лабиринтов. По заданным размерам ( количество строк и столбцов) построить прямоугольный лабиринт, который должен иметь единственное решение, все ячейки должны быть соединены с основным путем, дающим решение. Предусмотреть построение различных вариантов лабиринта при повторном обращении к программе пользователя.

15. Игра «пятнадцать». На квадратном поле размером 4х4 случайным образом расставлены 15 фишек с номерами от 1 до 15. Имеется одна свободная позиция. Расставить фишки по возрастанию их номеров так, как показано на рисунке 1 или рисунке 2. Передвигать фишки можно только на соседнюю свободную позицию.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 |  |

Рисунок 1 – Расставка фишки по возрастанию

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 15 | 14 |  |

Рисунок 2 – Расставка фишки по возрастанию

16. Игра «ханойская башня». Доска имеет три колышка. На первом нанизано n дисков убывающего вверх диаметра. Расположить диски в том же порядке на другом колышке. Диски можно перекладывать с колышка на колышек по одному. Класть больший диск на меньший не разрешается.

17. Игра «морской бой».

18. Игра «жизнь». Игра моделирует жизнь поколений колонии живых клеток, которые выживают, размножаются или погибают в соответствии со следующими правилами. Клетка выживает, если и только если она имеет двух или трех соседей из восьми возможных (рисунок 3а). Если у клетки только один сосед или вовсе ни одного, она погибает в изоляции (рисунок 3б). Если клетка имеет четырех или более соседей, она погибает от перенаселения (рисунок 3в). В любой пустой позиции, у которой ровно три соседа, в следующем поколении появляется новая клетка (рисунок 3г).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| х |  |  |  |  |  |  |  | х | х |  |  | х |  | х |
|  | х | х |  |  | х |  |  |  | х | х |  |  |  |  |
|  |  |  |  |  |  |  |  | х |  |  |  |  | х |  |

а. б. в. г.

Рисунок 3 – Моделирование жизни поколений колонии живых клеток

19. Игра «Великий комбинатор». Правила крайне просты. Один из игроков загадывает комбинацию из любых четырех цифр (0..9), не допускаются повторения, и цифра 0 не может быть первой. Второй игрок пытается раскрыть комбинацию, вводя четырехзначные числа (пробы). Первый сообщает, сколько цифр в пробе совпадает с комбинацией: сколько цифр совпадают по значению и по положению, и сколько входят в комбинацию, но стоят на другом месте. Один из игроков – компьютер.

20. Демонстрационная модель солнечной системы. Необходимо изобразить на экране компьютера Солнце и девять планет солнечной системы (от Меркурия до Плутона), а также основные их спутники (например, для Земли - Луну, для Марса - Фобос и Деймос), в их движении по орбитам. Можно считать, что вращение планет вокруг Солнца происходит в одной плоскости (поскольку плоскости орбит планет близки к плоскости земной орбиты), к этой плоскости можно отнести и орбиты спутников планет. Вращение планет вокруг своей оси можно не учитывать. Требуется также смоделировать и показать одно из астрономических событий в солнечной системе: пролет кометы через солнечную систему, запуск с какой-либо планеты искусственного спутника, перелет космического корабля с планеты на планету или другое.

21. Моделирование движения на автостраде. Рассматривается движение автомобилей на прямом однорядном (однополосном) участке автострады длиной 10 километров, без перекрестков, в течение некоторого времени. Автомобили появляются на одном конце дороги и проезжают по ней до другого конца, стараясь по возможности сохранить начальную, т.е. заданную при их появлении скорость. Разные автомашины могут иметь разную начальную скорость: начальная скорость - случайная величина, изменяющаяся в заданном диапазоне (например, от 50 до 100 км/час). Интервалы между появлениями автомобилей на дороге также являются случайными величинами из определенного интервала (например, от 1 до 5 секунд). Считается, что минимальное допустимое сближение двух автомобилей составляет одну длину (корпус) автомобиля, в ином случае происходит авария. Когда автомобиль приближается к идущей впереди машине на утроенное допустимое расстояние, он начинает притормаживать по определенному закону, пока его скорость не сравняется со скоростью передней машины.

22. Модельная система дешифрования текста. Данная система должна решать следующую задачу расшифровки текста, состоящего из 3-8 предложений на одном из естественных языков - русском, английском или другом. Дан зашифрованный текст, известно также, что для шифровки был использован метод подстановки, заключающийся в замене одних букв алфавита на другие, причем исключен простейший частный случай этого метода - циклический сдвиг алфавита. Необходимо расшифровать текст и найти примененный при шифровке шифр (т.е. набор пар буква - буква, число этих пар может быть несколько меньше количества букв в алфавите - если в расшифрованном тексте был использован не весь алфавит). Отметим, что указанный метод шифровки сохраняет структуру предложений текста -количество и длину входящих слов, расположение знаков препинания, в этом состоит его отличие от более сложных способов шифрования - например, с использованием ключевого слова. Расшифровка текста должна быть основана на знаниях определенных закономерностей, свойственных текстам на выбранном естественном языке -знаний о лексических, морфологических и грамматических особенностях этого языка.

[**Лабораторная работа №4. Разработка спецификации программы**](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\КР3.htm)

***Цель:*** Анализ требований и определение Спецификации программного обеспечения приложения при структурном подходе.

***Задание к лабораторной работе***

1. Разработать спецификацию ПС (функциональную и эксплуатационную часть), согласно теме индивидуального варианта лабораторной работы №3.

***Контрольные вопросы и задания СРС***

1. В чем сущность структурного подхода к программированию? Какие этапы охватывает данный подход?
2. Что понимают под термином «спецификации»? В чем сложность их уточнения? Назовите модели, используемые в качестве функциональных спецификаций при структурном подходе. Какие характеристики проектируемого программного обеспечения описывает каждая из них?

***Краткий методический материал***

Разработка любого программного обеспечения начинается с анализа требований к будущему программному продукту. В результате анализа получают специ­фикации разрабатываемого программного обеспечения: выполняют декомпозицию и со­держательную постановку решаемых задач, уточняют их взаимодействие и эксплуатаци­онные ограничения. В целом в процессе определения спецификаций строят общую мо­дель предметной области, как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, и кон­кретизируют его основные функции.

**1. Спецификации программного обеспечения при структурном подходе**

Как уже упоминалось в § 1.4, спецификации представляют собой полное и точное описание функций и ограничений разрабатываемого программного обеспечения. При этом одна часть спецификаций {функциональные) описы­вает функции разрабатываемого программного обеспечения, а другая часть {эксплуатационные) определяет требования к техническим средствам, на­дежности, информационной безопасности и т. д.

Определение отражает главные требования к спецификациям. Примени­тельно к функциональным спецификациям подразумевается, что:

требование ***полноты*** означает, что спецификации должны содержать всю существенную информацию, где ничего важного не было бы упущено, и отсутствует несущественная информация, например детали реализации, чтобы не препятствовать разработчику в выборе наиболее эффективных решений;

требование ***точности*** означает, что спецификации должны однозначно восприниматься как заказчиком, так и разработчиком.

Последнее требование выполнить достаточно сложно в силу того, что естественный язык для описания спецификаций не подходит: даже подробные спецификации на естественном языке не обеспечивают необходимой точности. Точные спецификации можно определить, только разработав некоторую ***формальную модель*** разрабатываемого программного обеспечения.

Формальные модели, используемые на этапе определения спецификаций можно разделить на две группы: модели, *зависящие от подхода к разработке* (структурного или объектно-ориентированного), и модели, *не зависящие* от него. Так диаграммы переходов состояний, которые демонстрируют особенности поведения разрабатываемого программного обеспечения при получении тех или иных сигналов извне, и математические моде­ли предметной области используют при любом подходе к разработке.

В рамках структурного подхода на этапе анализа и определения спецификаций используют три типа моделей: ориентированные на функции, ори­ентированные на данные и ориентированные на потоки данных. Каждую мо­дель целесообразно использовать для своего специфического класса программных разработок.

Следует иметь в виду, что все функциональные спецификации описывают одни и те же характеристики разрабатываемого программного обеспечения: перечень функций и состав обрабатываемых данных. Они различаются только системой приоритетов (акцентов), которая используется разработчи­ком в процессе анализа требований и определения спецификаций. Диаграммы переходов состояний определяют основные аспекты поведения про­граммного обеспечения во времени, диаграммы потоков данных - направление и структуру потоков данных, а концептуальные диаграммы классов - отношение между основными понятиями предметной области.

Поскольку разные модели описывают проектируемое программное обеспечение с разных сторон, рекомендуется использовать сразу несколько моделей и сопровождать их текстами: словарями, описаниями и т. п., кото­рые поясняют соответствующие диаграммы.

Так методологии структурного анализа и проектирования, основанные на моделировании потоков данных, обычно используют комплексное пред­ставление проектируемого программного обеспечения в виде совокупности моделей:

– диаграмм потоков данных (DFD - Data Flow Diagrams), описывающих взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;

– диаграмм «сущность-связь» (ERD - Entity-Relationship Diagrams), описывающих базы данных разрабатываемой системы;

диаграмм переходов состояний (STD - State Transition Diagrams), характеризующих поведение системы во времени;

спецификаций процессов;

словаря терминов.

Взаимосвязь элементов такой обобщенной модели показана на рис. 2.

***Спецификации процессов***. Спецификации процессов обычно представляют в виде краткого текстового описания, схем алгоритмов, псевдокодов, Flow-форм или диаграмм Насси-Шнейдермана. Поскольку описание процесса должно быть кратким и понятным как разработчику, так и заказчи­ку, для их спецификации чаще всего используют псевдокоды.

***Словарь терминов.*** Словарь терминов представляет собой краткое опи­сание основных понятий, используемых при составлении спецификаций. Он должен включать определение основных понятий предметной области, описание структур элементов данных, их типов и форматов, а также всех со­кращений и условных обозначений. Он предназначен для повышения степе­ни понимания предметной области и исключения риска возникновения раз­ногласий при обсуждении моделей между заказчиками и разработчиками.

В качестве примера приведем описание одного из терминов системы ре­шения комбинаторно-оптимизационных задач:

Термин ………. Алгоритм

Категория……. Понятие предметной области

Описание…….. В настоящем проекте используется для обозначения обоб­щенного понятия «реализация процедуры решения конкретной задачи выбранным методом»

Кроме указанных моделей в состав полной спецификации при любом подходе могут входить математические модели описания объектов предмет­ной области, которые позволяют уточнить основные соотношения анализи­руемых величин и накладываемые на них ограничения. Перейдем к более подробному рассмотрению перечисленных моделей.

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

[**Лабораторная работа №5. Анализ требований и определение спецификации программного обеспечения. Построение моделей проекта**](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\КР3.htm)

***Цель:*** Построение моделей по анализу и определению спецификации требований к программному обеспечению приложения.

***Задание к лабораторной работе***

Разработать согласно теме индивидуального варианта лабораторной работы №3:

– диаграмму переходов состояний,

– функциональную диаграмму (IDEF0, SADT-методологии),

– диаграмму потоков данных (DFD-методологии),

– диаграмму потоков данных, дополненной диаграммой потоков управляющих данных,

с использованием одного из автоматизированного средства  поддержки: CA ERwin Process Modeler, Design/IDEF, ARIS, ORACLE Designer

***Контрольные вопросы и задания СРС***

1. В каких случаях целесообразно использовать диаграммы переходов состояний? Разработайте диаграмму переходов для калькулятора, техническое задание на который составлялось вами в соответствии заданием к предыдущей главе.
2. В чем заключается основное различие между функциональными диаграммами и диаграммами потоков данных? Постройте оба вида диаграмм для выполнения вычислений с использованием внутренней памяти калькулятора. Проанализируйте сходство и различие. В каких случаях использование диаграмм потоков данных является предпочтительным?
3. Что называют «структурами данных»? Какие данные имеются в виду? В каких случаях структуры данных необходимо описывать? Какие модели используют для описания структур данных?
4. Опишите стек и очередь с использованием предлагаемых моделей описания данных. Какие аспекты этих структур остались не описанными и почему?
5. В каких случаях используют математические модели? Что понимают под адекватностью модели? Зачем необходимо выполнять доказательство адекватности и как строятся подобные доказательства?
6. Для чего строят структурные карты Константайна? Постройте структурные карты Константайна для задания
7. Чем структурные карты Джексона отличаются от структурных карт Константайна?
8. Что положено в основу методик Джексона и Варнье-Орра? Чем различаются данные методики?
9. Какие вопросы решают при проектировании структур данных? Какие характеристики проектируемых структур при этом учитывают? Предложите несколько вариантов структур данных дня программы задания 3. Какая из них является лучшей и почему?

***Краткий методический материал***

**1.Диаграммы переходов состояний**

Диаграмма переходов состояний является графической формой предо­ставления конечного автомата - математической абстракции, используемой для моделирования детерминированного поведения технических объектов или объектов реального мира.

На этапе анализа требований и определения спецификаций диаграмма переходов состояний демонстрирует поведение разрабатываемой программ­ной системы при получении управляющих воздействий. Под управляющими воздействиями или сигналами в данном случае понимают управляющую ин­формацию, получаемую системой извне. Например, управляющими воздей­ствиями считают команды пользователя и сигналы датчиков, подключенных к компьютерной системе. Получив такое управляющее воздействие, разраба­тываемая система должна выполнить определенные действия и или остаться в том же состоянии, или перейти в другое состояние взаимодействия с внеш­ней средой.

**2. Диаграммы потоков данных**

*Диаграммы потоков данных* позволяют специфицировать как функции разрабатываемого программного обеспечения, так и обрабатываемые им данные. При использовании этой модели систему представляют в виде ие­рархии диаграмм потоков данных, описывающих асинхронный процесс пре­образования информации с момента ввода в систему до выдачи пользовате­лю. На каждом следующем уровне иерархии происходит уточнение процес­сов, пока очередной процесс не будет признан элементарным.

**Примечание.** Модели потоков данных были независимо предложены сначала Е. Иорда­ном (1975). затем Ч. Геймом и Т. Сарсоном (1979). На этих моделях основаны классические методологии структурного анализа и проектирования программного обеспечения соответст­венно Йордана-Де Марка и Гейна-Сарсона. Та же модель используется в методологии стр\ к-турного анализа и проектирования SSADM (Structured Systems Analysis and Design Method). принятой в Великобритании в качестве национального стандарта разработки информацион­ных систем.

В основе модели лежат понятия внешней сущности, процесса, хранили­ща (накопителя) данных и потока данных.

*Внешняя сущность* - материальный объект или физическое лицо, вы­ступающие в качестве источников или приемников информации, например. заказчики, персонал, поставщики, клиенты, банк и т. п.

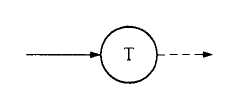
*Процесс* - преобразование входных потоков данных в выходные в соот­ветствии с определенным алгоритмом. Каждый процесс в системе имеет свой номер и связан с исполнителем, который осуществляет данное преобра­зование. Как в случае функциональных диаграмм, физически преобразова­ние может осуществляться компьютерами, вручную или специальными уст­ройствами. На верхних уровнях иерархии, когда процессы еще не определе­ны, вместо понятия «процесс» используют понятия «система» и «подсисте­ма», которые обозначают соответственно систему в целом или ее функцио­нально законченную часть.

*Хранилище данных* - абстрактное устройство для хранения информа­ции. Тип устройства и способы помещения, извлечения и хранения для тако­го устройства не детализируют. Физически это может быть база данных, файл, таблица в оперативной памяти, картотека на бумаге и т. п.

*Поток данных* - процесс передачи некоторой информации от источника к приемнику. Физически процесс передачи информации может происходить по кабелям под управлением программы или программной системы или вручную при участии устройств или людей вне проектируемой системы.

**3. Моделирование управляющих процессов с помощью диаграмм потоков данных**. Для представления управляющих процессов в проектируе­мых системах можно применить диаграммы переходов состояний, или диаграммы управляющих потоков данных, которые ис­пользуют понятия: управляющий процесс, управляющий поток данных и, возможно, хранилище управляющих данных.

Управляющий процесс получает с помощью управляющих потоков некоторую информацию о ситуации в системе и инициирует посредством управляющего потока соответствующие процессы.



Узел изменения типа потока данных

На диаграммах управляющих потоков данных используют те же обозначения, что и для обычных потоков, но изображают их пунктирной линией. Дополнительно может быть указан тип управляющего потока:

Т-поток (Trigger Flow - тригерный поток) - поток управления, который может только «включать» процесс - следующий управляющий сигнал опять «включит» процесс, даже если процесс уже активен;

А-поток (Activator Flow — активирующий поток) — поток управления, который может как «включать», так и «выключать» управляемый процесс - если процесс включен, то следующий сигнал его выключит;

E/D-поток (Enable/Disable Flow - переключающий поток) - поток управления, который может включать процесс сигналом по одной (Е) линии и выключать - сигналом по другой (D) линии.

При необходимости тип потока данных (управляющий или обычный) можно изменять. Для этого используют специальное обозначение - узел из­менения типа потока данных. К этому узлу поток подходит как поток данных, а выходит из него как управляющий поток.

**4.Диаграммы Джексона**. В основе диаграмм Джексона лежит предполо­жение о том, что структуры данных, так же, как и программ, можно строить из элементов с использованием всего трех основных конструкций: последо­вательности, выбора и повторения.

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

**Лабораторная работа № 6 «Визуализация данных двумерной и трёхмерной графикой»**

***Цель работы*:** приобретение практических навыков по построению графиков с помощью Модуля matplotlib.

**Задание к лабораторной работе**

1. Постройте график функции (**Приложение 1**)

2. Для построенного в рамках Части 1 графика измените:

• цвет линии;

• тип линии и маркеров;

• шаг выборки данных.

• Введите сетку.

• Сохраните полученный график в файл, опробуйте сохранять файл в разных форматах: png, pdf, jpg, eps

3. Постройте семейство функций на одном графике различными цветами:

– степенные многочлены с целыми степенями от 1 до 6 на отрезке [−1;1];

2. синусоиды y = sin(ωt) с частотами ω = 2π, ω = 3π, ..., ω = 8π на отрезке t ∈ [−1;1];

– синусоиды y = sin(2πt + φ0) с начальными фазами φ0 = 0, φ0 = π/6, ..., φ0 = 5π/6 на отрезке t ∈ [−1;1];

– логарифмические функции log2(x), ln(x) и log10(x) на отрезке x ∈ [1;10];

– гиперболические функции sh(x), ch(x) и th(x) на отрезке x ∈ [−10;10], для их вычислени

я воспользуйтесь их выражением через экспоненту.

4. Для построенного в пункте 3 графика

Сделайте сетку и легенду.

Перестройте графики так, чтобы каждая кривая располагалась на одном графике с помощью команды *subplot*, легенду уберите, а её текст переместите в название соответствующего графика.

Графики расположите на полотне:

• в одни столбец;

• в два столбца;

• в 3 столбца;

• в одну строку.

Перестройте графики из задания каждый в своём окне. Сделайте так, чтобы эти графики автоматически сохранялись каждый в свой файл.

5. Постройте круговую диаграмму, которая показывала бы доли от общего числа студентов вашей группы, сдавших сессию на:

– одни пятёрки,

– пятёрки и четвёрки,

– с тройками, но без задолжностей,

– с задолжностями, сумевших в итоге пересдать,

– несдавших и отчисленных (если такие имеются).

6. Постройте закрашенную контурную диаграмму и трёхмерный график для функций двух переменных, определённых в прямоугольной области x ∈ [−3;3], y ∈ [−3;3] (**Приложение 2**)

***Методический материал:***

Построение графиков — один из главных этапов обработки данных. Все современные компьютерные программы, предоставляющие функцию построения графиков, условно можно разделить на две категории: программы с визуальным интерфейсом, где построение и редактирование графика осуществляется средствами разного рода меню, полей ввода, лист-боксов, чек-боксов и других виджетов, и программы, где для построения графика необходимо писать команды, объединяемые в так называемы скрипты. К первой категории относятся, например, *Origin, MS Excel, OpenOﬃce/LibreOﬃce Calc, Statistica, Graphe*r, ко второй *– gnuplot*, многие математические пакеты, например, *MATLAB* и *SciLab* и различные библиотеки вроде *PGPlot* и *PLPlot*, имеющие поддержку во многих языках программирования.

Основное преимущество скриптового способа построения графика в том, что вы можете встроить его без проблем в вашу программу, производящую вычисления. Кроме того, скрипты позволяют легко перестраивать графики с новыми данными, автоматизировать построение графиков, дают почти неограниченный контроль над точностью позиционирования и размером деталей. Модуль *matplotlib* – специализированная библиотека для языка *Python*. Хотя основное её преимущество в простоте и быстроте использования, она позволяет делать графики очень высокого типографского качества. Модуль *matplotlib* базируется на возможностях *numpy*, установка которого обязательна для функционирования *matplotlib*.

Список рекомендуемой литературы

1. Сысоев М.В. «[Программирование для "нормальных" с нуля на языке Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Python_Москва_2018.pdf)», см. Глава 6.
2. Марк Лутц «[Программирование на Python](file:///D:\diskd_Учеба\!_УМК\!_УМКД_бакал_2021_22уч.г\ВТ_Современные%20методы%20разработки%20ПО_5%20сем\Literatura\Лутц_ОПП_Граф_интерфейсы_2.pdf)». Построение графических интерфейсов.
3. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

**Индивидуальные варианты заданий**

**Приложение 1**

|  |  |
| --- | --- |
| **№** | **Функция** |
|  | **x2** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **x3** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **X4** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **cos(2πt)** на отрезке t ∈ [−10;10] с шагом 1 и с шагом 0.25 |
|  | **1/t cos(2πt)** на отрезке t ∈ [1;10] с шагом 1 и с шагом 0.25 |
|  | **e−t cos(2πt)** на отрезке t ∈ [−10;10] с шагом 1 и с шагом 0.25 |
|  | **4sin(πt + π/8)−1** на отрезке t ∈ [−10;10] с шагом 1 и с шагом 0.25 |
|  | **2cos(t−2) + sin(2∗t−4)** на отрезке t ∈ [−20π;10π] с шагом π и с шагом π/12 |
|  | **ln(x + 1)** на отрезке x ∈ [0;e−1] с шагом 0.01 и с шагом 0.001 |
|  | **log2(|x|)** на отрезке x ∈ [−4;4] за исключением точки x = 0 с шагом 0.1 и с шагом 0.25 |
|  | **2x** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **ex** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **2−x** на отрезке x ∈ [−2;2] с шагом 0.01, с шагом 0.1, с шагом 0.25 |
|  | **x1/3** на отрезке x ∈ [1;125] с шагом 1 и с шагом 5, но так, чтобы значения 1 и 5 присутствовали среди аргументов |
|  | **x1/5** на отрезке x ∈ [1;32] с шагом 1 и с шагом 0.25 |

**Приложение 2**

|  |  |
| --- | --- |
| № | Функция |
|  | **z = x2 + y2** |
|  | **z = x2 −y2** |
|  | **z = x3 + y3** |
|  | **z = x3 −y3** |
|  | **z = x2 −y2 + x** |
|  | **z = x2 −y2 + y** |
|  | **z = x2 + y2 + x** |
|  | **z = x2 + y2 + y** |
|  | **z = sin(xy)** |
|  | **z = cos(xy)** |
|  | **z = cos(xy)** |
|  | **z = xy** |
|  | **z = x−sin(xy)** |
|  | **z = x + cos(xy)** |
|  | **z =(x2 + y2)1/2** |

**Лабораторная работа № 7 «Разработка интерфейса пользователя программного обеспечения»**

**Цель работы:** приобретение практических навыков по разработке пользовательских интерфейса для прикладного приложения.

**Задание к лабораторной работе**

Разработать проект пользовательского интерфейса к приложению, согласно теме индивидуального варианта лабораторной работы №3:

* Граф диалога пользовательского интерфейса
* Команды меню (диалога)
* Внешний вид пользовательского интерфейса
* Процесс изменения пользовательского интерфейса

**Контрольные вопросы и задания СРС**

1. Назовите основные типы интерфейсов. Чем характеризуется каждый из них? Какими средствами реализуется? Какие типы интерфейсов являются основными в наше время?
2. Перечислите психофизические особенности человека, которые необходимо учитывать при проектировании интерфейсов. Какие ограничения это накладывает на интерфейс?
3. Что понимают под термином «диалог»? Сколько диалогов может реализовывать программное обеспечение?
4. Назовите основные типы диалога и его формы. Какие модели используют для описания диалогов? Что служит исходными данными для проектирования диалогов?
5. Постройте граф диалога для простейшего графического редактора. Почему он имеет такой вид? В каких ситуациях граф диалога имеет вид цепи или дерева?
6. Предложите меню графического редактора. Сравните это меню с меню известных вам графических редакторов. Проанализируйте отличия.
7. Перечислите основные компоненты графических пользовательских интерфейсов. В каких случаях используют каждый из них?
8. Предложите интерфейс прямого манипулирования для графического редактора. В чем состоит основная сложность проектирования таких интерфейсов? В каких случаях их целесообразно использовать?
9. Какие интеллектуальные компоненты пользовательских интерфейсов существуют в настоящее время? Каковы их основные назначения? В каких случаях их целесообразно применять?

**Краткий методический материал**

***Этапы разработки пользовательского интерфейса***

Разработка пользовательского интерфейса включает те же основные этапы, что и разработка программного обеспечения:

* Постановка задачи - определение типа интерфейса и общих требований к нему;
* Анализ требований и определение спецификаций - определение сценариев использования и пользовательской модели интерфейса;
* Проектирование - проектирование диалогов и их реализация в виде процессов ввода-вывода;
* Реализация - программирование и тестирование интерфейсных процессов.

***Процесс проектирования пользовательского интерфейса***

На этапе проектирования пользовательского интерфейса разрабатывают следующие три модели пользовательского интерфейса:

* Модель программиста,
* Модель пользователя,
* Программную модель.

***1 Типы пользовательских интерфейсов и этапы их разработки***

*Пользовательский интерфейс* представляет собой совокупность про­граммных и аппаратных средств, обеспечивающих взаимодействие пользо­вателя с компьютером. Основу такого взаимодействия составляют диалоги. Под *диалогом* понимают *регламентированный* обмен ин­формацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной зада­чи: обмен информацией и координация действий. Каждый диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера.

Обмен информацией осуществляется передачей сообщений и управляющих сигналов. Различают:

• *входные сообщения*, которые генерируются человеком с помощью средств ввода: клавиатуры, манипуляторов, например мыши и т. п.;

• *выходные сообщения*, которые генерируются компьютером в виде текстов, звуковых сигналов и/или изображений и выводятся пользователю на экран монитора или другие устройства вывода информации.

 В основном пользователь генерирует сообщения следующих типов: запрос информации, запрос помощи, запрос операции или функции, ввод или изменение информации, выбор поля кадра и т. д. В ответ он получает: подсказки или справки, информационные сообщения, не требующие ответа, приказы, требующие действий, сообщения об ошибках, нуждающиеся в ответных действиях, изменение формата кадра и т.д.

*Типы интерфейсов*. По аналогии с процедурным и объектным подходом к программированию различают процедурно-ориентированный и объектно-ориентированный подходы к разработке интерфейсов (рисунок 4.2). Процедурно-ориентированные интерфейсы используют традиционную модель взаимодействия с пользователем, основанную на понятиях «процедура» и «операция». В рамках этой модели программное обеспечение предоставляет пользователю возможность выполнения некоторых *действий,* для которых пользователь определяет соответствующие данные и следствием выполнения которых является получение желаемых результатов.

Объектно-ориентированные интерфейсы используют модель взаимодействия с пользователем, ориентированную на манипулирование *объектами* предметной области. В рамках этой модели пользователю предоставляется возможность напрямую взаимодействовать с каждым объектом и инициировать выполнение операций, в процессе которых взаимодействуют несколько объектов. Пользователю предоставляется возможность создавать объекты, изменить их параметры и связи с другими объектами, а также инициировать взаимодействие этих объектов.

Применение процедурно-ориентированных интерфейсов в данном случае не означает использования структурного подхода к разработке соответствующего программного обеспечения. Более того, реализация современного процедурно-ориентированного пользовательского интерфейса на базе структурного подхода является очень сложной и трудоемкой задачей.

В таблице 1 перечислены основные отличия пользовательских моделей интерфейсов процедурного и объектно-ориентированного типов.

Таблица 1 – Отличия пользовательских моделей пользовательских интерфейсов

|  |  |
| --- | --- |
| Процедурно-ориентированные пользовательские интерфейсы | Объектно-ориентированные пользовательские интерфейсы |
| Обеспечивают пользователей функциями, необходимыми для выполнения задач | Обеспечивают пользователям возможность взаимодействия с объектами |
| Акцент делается на задачи | Акцент делается на вводные данные и результаты |
| Пиктограммы представляют приложе­ния, окна или операции | Пиктограммы представляют объекты |
| Содержание папок и справочников отображается с помощью таблиц и списков | Папки и справочники являются визуальными контейнерами объектов |

Различают процедурно-ориентированные интерфейсы трех типов: «примитивные», меню и со свободной навигацией.

*Примитивным* называют интерфейс, который организует взаимодействие с пользователем в консольном режиме. Обычно такой интерфейс реализует конкретный сценарий работы программного обеспечения, например: ввод данных - решение задачи - вывод результата

*Интерфейс-меню* в отличие от примитивного интерфейса позволяет пользователю выбирать необходимые операции из специального списка, выводимого ему программой. Эти интерфейсы предполагают реализацию множества сценариев работы, последовательность действий, которых определяется пользователем.

Различают одноуровневые и иерархические меню. Первые используют для сравнительно простого управления вычислительным процессом, когда вариантов немного (не более 5-7), и они включают операции одного типа, например, Создать, Открыть, Закрыть и т. н. Вторые - при большом количестве вариантов или их очевидных различиях, например, операции с файлами и операции с данными, хранящимися в этих файлах.

Интерфейс-меню предполагает, что программа находится либо в состоя­нии Уровень меню, либо в состоянии Выполнение операции. В состоянии Уровень меню осуществляется вывод меню соответствующего уровня и выбор нужного пункта меню, а в состоянии Выполнение операции реализуется сценарии выбранной операции. В порядке исключения иногда пользователю предоставляется возможность завершения операции независимо от стадии выполнения сценария и/или программы, например, по нажатию клавиши Esc.

Древовидная организация меню предполагает строго ограниченную навигацию: либо переходы «вверх» к корню дерева, либо - «вниз» по выбранной ветви. Каждому уровню иерархического меню соответствует свое определенное окно, содержащее пункты данного уровня. При этом возможны два варианта реализации меню: каждое окно меню занимает весь экран или на экране одновременно присутствуют несколько меню разных уровней. Во втором случае окна меню появляются при выборе пунктов соответствующе­го верхнего уровня — «выпадающие» меню.

*Интерфейсы со свободной навигацией* также называют *графическими  пользовательскими  интерфейсами* (GUI - Graphic User Interlace) или интерфейсами WYSIWYG (What You See Is What You Get - что видишь, то и получишь), т.е., что пользователь видит на экране, то он и получит при печати.

Графические интерфейсы поддерживают концепцию интерактивного взаимодействия с программным обеспечением, осуществляя визуальную обратную связь с пользователем и возможность прямого манипулирования объектами и информацией на экране.

В отличие от интерфейса-меню интерфейс со свободной навигацией обеспечивает возможности осуществления любых допустимых в конкретном состоянии операций, доступ к которым возможен через различные интерфейсные компоненты. Например, окна программ обычно содержат:

• меню различных типов: ниспадающее, кнопочное, контекстное;

• разного рода компоненты ввода данных.

Причем выбор следующей операции в меню осуществляется как мышью, так и с помощью клавиатуры.

Существенной особенностью интерфейсов данного типа является способность изменяться в процессе взаимодействия с пользователем, предлагая выбор только тех операций, которые имеют смысл вконкретной ситуации.

Объектно-ориентированные интерфейсы пока представлены только *интерфейсом прямого манипулирования.* Этот тип интерфейса предполагает, что взаимодействие пользователя с программным обеспечением осуществляется посредством выбора и перемещения *пиктограмм,* соответствующих объектам предметной области.

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

**Лабораторная работа №8. Разработка технического проекта программного обеспечения**

**Цель:** Проектирование приложения при структурном подходе. Приобретение практических навыков использованию псевдокода, структурных карт Константайна, иерархических схем Джексона

***Задание к лабораторной работе***

Разработать проект Программного приложения согласно индивидуальному варианту лабораторной работы №3:

– Разработать алгоритм программы с использованием метода пошаговая детализация (псевдокода),

– Разработать структурную схему Приложения,

– Разработать архитектуру модулей кода Приложения с использованием структурных карт Константайна

– Разработать структуру программы (иерархические схемы Джексона), основанную на декомпозиции данных

***Контрольные вопросы и задания СРС***

* 1. Как описывают структуру программного обеспечения при объектном подходе? Что такое «пакет»? Для чего используют диаграммы пакетов? 2.Какие стереотипы классов введены и почему?

1. Разработайте диаграмму пакетов графического редактора, описанного вами при выполнении задания 9 к гл. 6. Какие пакеты включены в эту диаграмму и почему? Какие пакеты будут связаны между собой?
2. Постройте диаграмму последовательности действий для объектов любых предложенных вами пакетов. Какими сообщениями обмениваются объекты? Какую информацию программист получит; анализируя эту диаграмму?
3. Какую диаграмму используют при уточнении взаимодействия объектов? Постройте эту диаграмму для объектов предыдущего задания.
4. Перечислите основные компоненты классов. Как описывают эти компоненты?
5. В каких случаях используют диаграммы состояний объекта? Построите диаграмму состояний для любого управляющего объекта.
6. Постройте уточненную диаграмму классов по результатам исследования взаимодействия объектов. Какая еще информация необходима для реализации этих классов?
7. Что понимают под диаграммой компонентов? Какую информацию она содержит? В каких случаях целесообразно строить диаграммы компонентов?
8. Какую информацию содержит диаграмма размещения? В каких случаях целесообразно использовать эти диаграммы?

**Краткий методический материал**

Сущность структурного подхода заключается в деком­позиции программы или программной системы по функциональному принципу. Все предлагаемые методы декомпозиции используют интерфейсы простейшего типа: прими­тивные интерфейсы и традиционнее меню, и рассчитаны на анализ и проектирование как структур данных, так и обрабатывающих их программ.

Причем в большинстве случаев первичным считают проектирование обрабатыва­ющих компонентов, проектирование же структур данных выполняют параллельно. Су­ществует и альтернативный подход, при котором первичным считают проектирование данных, а обрабатывающие программы получают, анализируя полученные структуры данных.

В любом случае проектирование программного обеспечения начинают с определе­ния его структуры.

1. Разработка структурной и функциональной схем

Процесс проектирования сложного программного обеспечения начина­ют с уточнения его структуры, т. е. определения структурных компонентов и связей между ними. Результат уточнения структуры может быть представлен в виде структурной и/или функциональной схем и описания (спецификации) компонентов.

Структурная схема разрабатываемого программного обеспечения. *Структурной* называют схему, отражающую *состав* и *взаимодействие по управлению* частей разрабатываемого программного обеспечения.

Самый простой вид программного обеспечения - программа, которая в качестве структурных компонентов может включать только подпрограммы и библиотеки ресурсов. Разработку структурной схемы программы обычно вы­полняют методом пошаговой детализации.

Структурными компонентами программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиоте­ки ресурсов и т. п.

Структурная схема программного комплекса демонстри­рует передачу управления от программы-диспетчера соответствующей про­грамме.

Структурная схема программной системы, как правило, по­казывает наличие подсистем или других структурных компонентов. В отли­чие от программного комплекса отдельные части (подсистемы) программной системы интенсивно обмениваются данными между собой и. возможно, с ос­новной программой. Структурная же схема программной системы этого обычно не показывает. Более полное представление о проектируемом программном обеспече­нии с точки зрения взаимодействия его компонентов между собой и с внеш­ней средой дает функциональная схема.

Функциональная схема. Функциональная схема или схема данных (ГОСТ 19.701-90) - схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функцио­нальных схем используют специальные обозначения, установленные стан­дартом. Основные обозначения схем данных по ГОСТ 19.701-90.

Все компоненты структурных и функциональных схем должны быть описаны. При структурном подходе особенно тщательно необходимо прора­батывать спецификации межпрограммных интерфейсов, так как от качества их описания зависит количество самых дорогостоящих ошибок. К самым до­рогим относятся ошибки, обнаруживаемые при комплексном тестировании, так как для их устранения могут потребоваться серьезные изменения уже от­лаженных текстов.

2. Использование метода пошаговой детализации для проектирования структуры программного обеспечения

Структурный подход к программированию в том виде, в котором он был сформулирован в 70-х годах XX в., предлагал осуществлять декомпозицию программ методом пошаговой детализации. Результатом декомпозиции является структурная схема программы, которая представляет собой многоуровневую иерархическую схему взаимодействия подпрограмм по управле­нию. Минимально такая схема отображает два уровня иерархии, т. е. показы­вает общую структуру программы. Однако тот же метод позволяет получить структурные схемы с большим количеством уровней.

Метод пошаговой детализации реализует нисходящий подход и базируется на основных конструкциях структурного программирования . Он предполагает пошаговую разработку алгоритма. Каж­дый шаг при этом включает разложение функции на подфункции. Так на пер­вом этапе описывают решение поставленной задачи, выделяя общие подза­дачи на следующем аналогично описывают решение подзадач, формулируя при этом подзадачи следующего уровня. Таким образом, на каждом шаге происходит уточнение функций проектируемого программного обеспечения. Процесс продолжают, пока не доходят до подзадач, алгоритмы решения ко­торых очевидны.

Декомпозируя программу методом пошаговой детализации, следует придерживаться основного правила структурной декомпозиции, следующего из принципа вертикальною управления: в первую очередь дета­лизировать управляющие процессы декомпозируемого компонента, оставляя уточнение операций с данными напоследок. Это связано с тем, что приори­тетная детализация управляющих процессов существенно упрощает струк­туру компонентов всех уровней иерархии и позволяет не отделять процесс принятия решения от его выполнения: так, определив условие выбора неко­торой альтернативы, сразу же вызывают модуль, ее реализующий.

Детализация операций со структурами в последнюю очередь позволит отложить уточнение их спецификаций и обеспечит возможность относитель­но безболезненной модификации этих структур за счет сокращения количе­ства модулей, зависящих от этих данных.

Кроме этого, целесообразно придерживаться следующих рекомендаций:

• не отделять операции инициализации и завершения от соответствую­щей обработки, так как модули инициализации и завершения имеют плохую связность (временную) и сильное сцепление (по управлению);

• не проектировать слишком специализированных или слишком универсальных модулей, так как проектирование излишне специальных модулей увеличивает их количество, а проектирование излишне универсальных мо­дулей повышает их сложность;

• избегать дублирования действий в различных модулях, так как при их изменении исправления придется вносить во все фрагменты программы, где они выполняются - в этом случае целесообразно просто реализовать эти действия в отдельном модуле;

• группировать сообщения об ошибках в один модуль по тину библиоте­ки ресурсов, тогда будет легче согласовать формулировки, избежать дублиро­вания сообщений, а также перевести сообщения на другой язык.

При этом, описывая решение каждой задачи, желательно использовать не более 1-2-х структурных управляющих конструкций, таких, как цикл-пока или ветвление, что позволяет четче представить себе структуру организу­емого вычислительного процесса.

3. Структурные карты Константайна

На структурной карте отношении между модулями представляют в виде графа, вершинам которого соответствуют модули и общие области данных, а дугам - межмодульные вызовы и обращения к общим областям данных.

Различают четыре типа вершин (рис. 7):

• модуль — подпрограмма,

• подсистема — прoграмма,

• библиотека - совокупность подпрограмм, размещенных в отдельном модуле,

• область данных специальным образом оформленная совокупность данных, к которой возможно обращение извне.

При этом отдельные части программной системы (программы, подпрограммы) могут вызываться последовательно, параллельно или как сопро­граммы (рис. 8).

Чаще всею используют *последовательный* вызов, при котором модули, передав управление, ожидают завершения выполнения вызванной програм­мы или подпрограммы, чтобы продолжить прерванную обработку.

4. Проектирование программного обеспечения, основанное на декомпозиции данных

Практически одновременно были предложены методики проектирования программного обеспечения Джексона и Варнье-Орра, основанные на декомпозиции данных. Обе методики предназначе­ны для создания «простых» программ, работающих со сложными, но иерар­хически организованными структурами данных. При необходимости разра­ботки программных систем в обоих случаях предлагается вначале разбить систему на отдельные программы, а затем использовать данные методики.

Методика Джексона. При создании своей методики М. Джексон исхо­дил из того, что структуры исходных данных и результатов определяют структуру программы.

Методика основана на поиске соответствий структур исходных данных и результатов. Однако при ее применении возможны ситуации, когда на ка­ких-то уровнях соответствия отсутствуют. Например, записи исходного фай­ла сортированы не в том порядке, в котором соответствующие строки долж­ны появляться в отчете. Такие ситуации были названы «столкновения ми». Выделяют несколько типов столкновений, которые разрешают по-разному. При различной последовательности записей их просто сортируют до обра­ботки. Более подробно способы разрешения столкновений изложены в [33|.

Разработка структуры программы в соответствии с методикой выполняется следующим образом:

• строят изображение структур входных и выходных данных;

• выполняют идентификацию связей обработки (соответствия) между этими данными;

• формируют структуру программы на основании структур данных и обнаруженных соответствий;

• добавляют блоки обработки элементов, для которых не обнаружены соответствия;

• анализируют и обрабатывают несоответствии, т.е. разрешают «столкновения»;

• добавляют необходимые операции (ввод, вывод, открытие/закрытие файлов и т. н.);

• записывают программу в структурной нотации (псевдокоде).

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

**Лабораторная работа № 9** **Разработка прикладного приложения**

**Цель:** Приобретение практических навыков по кодированию программного обеспечения

**Задание к лабораторной работее**

1. Выполнить кодирование программного обеспечения на Python, согласно индивидуальноuj варианта задания, для проекта разработанного в рамках лабораторных работах №3-5, №7-8.
2. Оформить нижеследующие документы приложения, согласно индивидуальному варианту задания:

– Лист задания

– Постановка задачи

– Проектный документ «Технические задание»

– Проектный документ «Спецификация требований», включающий:

– Словарь терминов

– Функциональные требования

– Нефункциональные требования

– Обратные требования

– Диаграмму переходов состояний

– Математическую модель предметной области

– Функциональные диаграммы

– Диаграммы потоков данных

– Диаграмму потоков данных, дополненную диаграммой потоков управляющих данных

– Структуру данных и диаграммы отношений компонентов данных

– Проектный документ «Технический проект», включающий:

– Результаты пошаговой детализации (псевдокод) при определении алгоритма работы и структурной схемы Программного приложения

– Структурная схема Программного приложения

– Функциональная схема Программного приложения

– Структурная карта отношения (карта Константайна) между программными модулями Программного приложения

– Структуры входных, выходных данных программы (на основе методики Джексона и Варнье-Орра)

– Граф диалога пользовательского интерфейса, Проект интерфейса пользователя, с обоснованием выбора его вида.

– Результаты тестирования:

– Программного обеспечения

– Пользовательского интерфейса

**Контрольные вопросы СРС**

1. Назовите основные виды программной документации. Охарактеризуйте каждый из них. В каких случаях их используют?

2. Что должно описываться в пояснительной записке? Кому она предназначена? Почему в пояснительной записке обычно описывают не только принятые решения, но и отвергнутые варианты?

3. На кого рассчитано руководство пользователя? Что оно должно содержать? В каких ситуациях вы читаете руководство пользователя? Вспомните прочитанные вами руководства пользователя. Что вам в них не понравилось?

**Список рекомендуемой литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

**Лабораторная работа № 10 «Тестирование и отладка программного обеспечения»**

**Цель работы:** приобретение практических навыков по тестированию программных продуктов

**Задание к лабораторной работе**

1. 1. Выполнить тестирование отдельных модулей (2-3 модуля) программы, созданной согласно теме индивидуального варианта лабораторной работы №3, с использованием структурного подхода. Подготовить отчет о покрытии, который представляет собой список структурных элементов покрываемого программного кода (функций, процедур или методов), содержащий для каждого структурного элемента следующую информацию:

* Название функции, процедуры или метода
* Тип покрытия
* Количество покрываемых элементов (строк, ветвей, логических условий)
* Степень покрытия (в процентах или в абсолютном выражении)
* Список непокрытых элементов (в виде участков непокрытого программного кода с номерами строк)

2. Для каждого поля ввода данных выполните следующее:

* Проанализируйте значения, которые в него можно вводить. Сгруппируйте их в классы.
* Проанализируйте возможные граничные условия. Их можно описать, исходя из определений классов, но возможно, что в ходе этого анализа добавятся и новые классы значений.
* Нарисуйте таблицу, в которой перечислите все классы значений для каждого поля ввода и все интересные тестовые примеры (граничные и другие особые значения).
* Протестируйте программу, используя записанные значения (а если их слишком много, то некоторое их подмножество). Протестировать программу, означает не только запустить ее, ввести данные и посмотреть, не произойдет ли сбой, – важно, чтобы программа правильно использовала введенные данные.

# Контрольные вопросы СРС

1. Какой процесс называют отладкой? В чем его сложность?
2. Назовите основные типы ошибок. Как они проявляются при выполнении программы?
3. Перечислите основные методы отладки. В чем заключается различие между ними? Возьмите любую программу, содержащую ошибки, и попробуйте найти ошибку, используя каждый из перечисленных методов. Какой метод для вас проще и естественней и почему?
4. Какие средства получения дополнительной информации об ошибках вы знаете? Вспомните, какие ошибки вы искали дольше всего и почему. В каких случаях дополнительная информация позволяет найти ошибку?

**Методический материал**

Тестирование – очень, важный и трудоемкий этап процесса разработки программного обеспечения, так как правильное тестирование позволяет выявить подавляющие большинство ошибок, допущенных при составлении программ.

Для повышения качества тестирования рекомендуется соблюдать следующие *основные принципы:*

* предполагаемые результаты должны быть известны до тестирования;
* необходимо досконально изучать результаты каждого теста;
* необходимо проверять действия программы на неверных данных;
* необходимо проверять программу на неожиданные побочные эффекты на неверных данных.

Следует также иметь в виду, что вероятность наличия необнаруженных ошибок в части программы пропорциональны количеству ошибок уже найденных в этой части.

Формирование тестовых наборов. *Удачным* следует считать тест, который обнаруживает хотя бы одну ошибку. С этой точки зрения необходимо использовать такие наборы тестов, каждый из которых с максимальной вероятностью может обнаружить ошибку.

Формирование набора тестов имеет большое значение, поскольку тестирование является одним из наиболее трудоемких этапов (от 30 до 60 % общей трудоемкости) создания программного продукта.

Существуют два принципиально различных подхода к формированию тестовых наборов: структурный и функциональный.

*Структурный подход* базируется на том, что *известна структура* тестируемого программного обеспечения, в том числе его алгоритмы («стеклян­ный ящик»). В этом случае тесты строят так, чтобы проверить правильность реализации заданной логики в коде программы.

*Функциональный* подход основывается на том, что структура программного обеспечения не известна («черный ящик»). В этом случае тесты строят, опираясь на функциональные спецификации. Этот подход называют также подходом, управляемым данными, так как при его использовании тесты строят на базе различных способов декомпозиции множества данных.

Наборы тестов, полученные в соответствии с методами этих подходов, обычно объединяют, обеспечивая всестороннее тестирование программного обеспечения,

Критерии завершения тестирования и отладки. Одним из самых сложных является вопрос о том, когда следует завершать тестирование, поскольку невозможно гарантировать, что в разрабатываемом программном обеспечении не осталось ошибок.

Все критерии можно разделить на три группы:

• основанные на методологиях проектирования тестов - определенное количество тестов, полученных по методам анализа причинно-следственных связей, анализа граничных значений и предположения об ошибке, перестают выявлять ошибки;

• основанные на оценке возможного количества ошибок - возможное ко­личество ошибок оценивают экспертно, или по специальным методикам, а затем завершают тестирование при нахождении примерно 93-95% ошибок;

• основанные на исследовании результатов тестирования - строят график зависимости количества обнаруженных ошибок or времени тестирования, если он напоминает график, представленный на рисунке 5.2, то тестирование можно завершать.

**Список литературы**

1. Электронный учебник по дисциплине «Технология программирования», КарТУ, 2020г.

2.Электронный учебник по дисциплине «Процессы разработки программ», КарТУ, 2015 г.

3. Солодовникова, И. В. Разработка программного обеспечения: учебное пособие для магистрантов и студентов / И. В. Солодовникова, Е. Г. Клюева. - Караганда: КарГТУ, 2017. - 95 с.

4. Марк Лутц. Программирование на Python. Тома 1 и 2, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с

5. Методы и технологии программирования: Утверждено Ученым советом университета в качестве учебного пособия / Д. К. Жумагулова, А. А. Калинин, Ш. З. Телбаева, А. М. Кочкин; Министерство образования и науки Республики Казахстан, Карагандинский государственный технический университет, Кафедра "Автоматизация производственных процессов". - Караганда: КарГТУ, 2020. - 44 с.

6. Парфилова, Н. И. Программирование: Основы алгоритмизации и программирования: учебник для студентов вузов - М.: Изд. центр "Академия", 2012. - 240 с. - (Высшее профессиональное образование. Бакалавриат).

7. Анашкина Н.В. Технологии и методы программирования: учебное пособие для студентов вузов, аспирантов и преподавателей / Н. В. Анашкина, Н. Н. Петухова, В. Ю. Смольянинов. - М.: Академия, 2012.

9. Зайцева, С. В. Проектирование информационных систем: учебное пособие для студентов / С. В. Зайцева, М. К. Баймульдин. - Караганда: КарГТУ, 2015. - 104 с.

1. Томилов, А. Н. Программное обеспечение ЭВМ. Базы данных и системы управления базами данных: учебное пособие для студентов / А. Н. Томилов, Л. К. Сулейменова, Н. И. Томилова. - Караганда: КарГТУ, 2016. - 73 с.
2. Анашкина Н.В. Технологии и методы программирования: учебное пособие для студентов вузов, аспирантов и преподавателей / Н. В. Анашкина, Н. Н. Петухова, В. Ю. Смольянинов. - М.: Академия, 2012.
3. [www.intuit.ru](http://www.intuit.ru) Язык программирования Python.
4. Видеолекции на Youtube (открытая библиотека видеолекций): <https://www.youtube.com/watch?v=xhoX3-NdM9k> .
5. [www.intuit.ru](http://www.intuit.ru) Язык программирования Python.
6. Доусон М. Программируем на Python. - СПб.: Питер, 2014. - 416 с.
7. Видеолекции на Youtube (открытая библиотека видеолекций): <https://www.youtube.com/watch?v=xhoX3-NdM9k> .
8. Рыбина Г.B. Основы построения интеллектуальных систем. Учебное пособие. –М.: Финансы и статистика, 2010. – 432 с.