**Лабораторная работа №16**

**Выбор и обоснование средств реализации проекта для разрабатываемой программной системы.**

**Цель работы:**

1. Изучить критерии выбора языка программирования. Научиться обосновывать выбор среды разработки в соответствии с критериями выбора языка программирования.
2. Изучить основные парадигмы программирования.
3. Изучить правила формирования листинга программы.

**Задание 1. Критерии выбора языка программирования.**

Критерии выбора языка программирования:

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

Для создания мобильного приложения по мой теме я буду использовать C#, потому что он является многоцелевым языком программирования, который может использоваться для создания мобильных приложений. Он является мультиплатформенным, что означает, что вы можете создавать приложения для разных платформ, таких как Android и iOS. C# также очень быстро развивается и имеет большое сообщество разработчиков, которые могут помочь вам с проблемами и вопросами. Кроме того, код на C# легко читается, что увеличивает вероятность успешного и быстрого обновления контента. В общем, использование C# для разработки мобильных приложений может быть хорошим выбором.

**Задание 2. Изучение парадигм программирования.**

Существует множество различных парадигм программирования, включая императивную, декларативную, объектно-ориентированную, функциональную, логическую и другие. Каждая парадигма имеет свои особенности и преимущества, и выбор парадигмы зависит от конкретной задачи и предпочтений программиста. Например, императивная парадигма фокусируется на последовательном выполнении команд, в то время как декларативная парадигма фокусируется на описании желаемого результата. Объектно-ориентированная парадигма фокусируется на создании объектов и взаимодействии между ними, в то время как функциональная парадигма фокусируется на использовании функций для преобразования данных.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Название парадигмы | Сущность парадигмы, основные идеи, принципы, объекты. | Языки поддерживающие данную парадигму.  (2-3 языка) | Достоинства | Недостатки | Примеры программ |
| 1. | Императивное программирование | использование именованных переменных; использование оператора присваивания; использование составных выражений; использование подпрограмм. | C, C++, C# | Простота, контроль, эффективность | Сложность, трудности с отладкой, низкая переносимость | Калькулятор, сортировка массива, игра «Змейка» |
| 2. | Декларативное программирование | Разработчику нужно сосредоточиться не на том, что нужно сделать, а на том, какой результат должен быть достигнут. Принципы парадигмы: высокий уровень абстракции и отделение логики программы от её реализации | SQL, HTML | Высокий уровень абстракции, надёжность и переносимость, параллелизм и оптимизация. | Сложность и неочевидность, ограниченность и неполнота, отсутствие контроля и обратной связи | Запрос к базе данных, веб-страница, экспертная система, модель физической системы. |
| 3. | Структурное программирование | Модульность, локализация, последовательность; переменные, функции и структуры данных. | C, Pascal | Читаемость и понятность, надёжность и отладка, модульность и повторное использование. | Ограниченность и неэффективность, сложность и избыточность, низкая абстракция и гибкость. | Факториал, Фибоначчи, бинарный поиск. |
| 4. | Процедурное программирование | Модульность, абстракция и повторное использование; переменные, функции и структуры данных. | C, Pascal, Basic | Эффективность и контроль, повторное использование и модульность, абстракция и понятность. | Сложность и избыточность, низкая гибкость и переменность, низкая надёжность и отладка. | Факториал,Фибоначчи, бинарный поиск. |
| 5. | Модульное программирование | Абстракция, инкапсуляция и связность; модули, переменные, функции и структуры данных. | Python, Ada | Надёжность и сопровождаемость, повторное использование и абстракция, масштабируемость и параллелизм. | Сложность и несовместимость, низкая эффективность и производительность, низкая гибкость и адаптивность. | Компилятор, операционная система, браузер. |
| 6. | Объектно-ориентированное программирование | Абстракция, инкапсуляция, наследование, полиморфизм; объект, класс, метод, свойство. | Java, Python, C++ | Повышение модульности и повторного тестирования, упрощение проектирования и тестирования, улучшение читаемости и поддержки кода. | Увеличение сложности кода, увеличение затрат на ресурсы, ухудшение производительности. | Microsoft Word, Adobe Photoshop… |
| 7. | Функциональное программирование | Чистота, неизменяемость, ссылочная прозрачность, высокий уровень абстракции. | Haskell, F# | Повышение надёжности и отладки кода, повышение модульности и композиции кода, повышение выразительности и абстракции кода, повышение эффективности и масштабируемости. | Снижение производительности и потребления памяти, снижение читаемости и понимания кода, снижение совместимости и интеграции кода, снижение доступности и обучаемости кода. | GNU Emacs, WhatsApp, Twitter. |
| 8. | Логическое программирование | Декларативность, реляционность, недетерминизм, рекурсия. | Mercury, Visual Prolog | Повышение надёжности и корректности кода, повышение модульности и повторного использования кода, повышение выразительности и абстракции кода, повышение эффективности и масштабируемости кода. | Снижение производительности и потребления памяти, снижение читаемости и понимания кода, снижение совместимости и интеграции кода, снижение доступности и обучаемости кода. | Экспертные системы, системы автоматического доказательства теорем, системы обработки естественного языка, системы искусственного интеллекта. |

Наиболее часто используется парадигма Объектно-ориентированного программирования, потому что её поддерживают большинство популярных языков.

Для своего индивидуального задания я буду использовать парадигму объектно-ориентированного программирования, потому что недостатков у неё меньше, чем в других парадигмах.

**Задание 3. Изучение правил Формирования листинга программ.**

**Структурное программирование:**

#include <stdio.h>

// Функция для вычисления факториала числа

int factorial(int n) {

int result = 1;

for (int i = 1; i <= n; i++) {

result \*= i;

}

return result;

}

// Процедура для вывода первых n чисел Фибоначчи

void fibonacci(int n) {

int a = 0, b = 1, c;

for (int i = 0; i < n; i++) {

printf("%d ", a);

c = a + b;

a = b;

b = c;

}

}

int main() {

int x;

printf("Enter a number: ");

scanf("%d", &x);

// Вызов функции

printf("Factorial of %d is %d\n", x, factorial(x));

// Вызов процедуры

printf("The first %d Fibonacci numbers are: ", x);

fibonacci(x);

printf("\n");

return 0;

}

**Модульное программирование:**

// Модуль math\_utils.h

#ifndef MATH\_UTILS\_H

#define MATH\_UTILS\_H

// Функция для вычисления факториала числа

int factorial(int n);

// Функция для вычисления n-го числа Фибоначчи

int fibonacci(int n);

#endif

// Модуль math\_utils.c

#include "math\_utils.h"

int factorial(int n) {

int result = 1;

for (int i = 1; i <= n; i++) {

result \*= i;

}

return result;

}

int fibonacci(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

return fibonacci(n-1) + fibonacci(n-2);

}

**Объектно-ориентированное программирование:**

#include <iostream>

#include <string>

class Person {

public:

// Конструктор

Person(std::string name, int age) {

this->name = name;

this->age = age;

}

// Метод для вывода информации о человеке

void display() {

std::cout << "Name: " << name << ", Age: " << age << std::endl;

}

private:

std::string name; // Имя

int age; // Возраст

};

int main() {

Person alice("Alice", 25);

alice.display(); // Выводит: Name: Alice, Age: 25

Person bob("Bob", 32);

bob.display(); // Выводит: Name: Bob, Age: 32

return 0;

}

**Контрольные вопросы:**

1. Язык программирования – это формальный язык, предназначенный для записи компьютерных программ. Он определяет набор лексических, синтаксических и семантических правил, которые определяют внешний вид программы и действия, которые выполнит исполнитель (обычно ЭВМ) под её управлением.

Алфавит языка программирования – это набор символов, которые используются в языке.

Синтаксис языка программирования – это набор правил, описывающих комбинации символов алфавита, которые считаются правильно структурированной программой или её фрагментом.

Семантика в программировании – это дисциплина, изучающая формализации значений конструкций языков программирования посредством построения их формальных математических моделей.

Стандарт языка программирования – это документ, который определяет синтаксис и семантику языка.

Существует множество классификаций языков программирования. Например, одна из классификаций разделяет языки на процедурные, объектно-ориентированные, функциональные и логические. Другая классификация разделяет языки на низкоуровневые и высокоуровневые. Ещё одна классификация разделяет языки на интерпретируемые и компилируемые.

1. Критерии выбора языка программирования могут включать в себя следующее:
   1. Сфера применения: каждый язык программирования имеет свою сферу применения и свой список связанных технологий (он называется «стек»). Например, для веб-разработки часто используются языки JavaScript, PHP, Python, Java, CSS и HTML.
   2. Типизация: типизация языка программирования определяет, как вы будете писать программы и как они затем будут работать. Типизация бывает явная или неявная, слабая или сильная и статическая или динамическая.
   3. Сложность изучения: некоторые языки программирования более просты для изучения, чем другие. Например, HTML и CSS очень простые и в основах можно разобраться буквально за пару дней.
   4. Сообщество: наличие большого и активного сообщества разработчиков может облегчить изучение языка программирования и помочь в решении проблем.
   5. Рынок труда: спрос на разработчиков определенного языка программирования может варьироваться в зависимости от региона и отрасли. Выбор языка программирования может зависеть от того, насколько легко найти работу с использованием этого языка.
2. Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ. Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Существует множество различных парадигм программирования, включая императивную, декларативную, процедурную, структурную, аспектно-ориентированную, объектно-ориентированную, агентно-ориентированную, компонентно-ориентированную, прототипно-ориентированную и другие.

Также существует понятие стандарта оформления кода (стиля программирования), который представляет собой набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

* 1. Императивное программирование – Это парадигма программирования, в которой в исходном коде программы записываются инструкции (команды), которые должны выполняться последовательно. Данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями.

Декларативное программирование – Это парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается ожидаемый результат, а не способ его получения.

Структурное программирование – Это парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков.

Процедурное программирование – Это 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.

Модульное программирование – Это подход к программированию, который подразумевает разделение программы на отдельные модули или компоненты. Каждый модуль выполняет определенную функцию и может быть разработан независимо от других модулей.

Объектно-ориентированное программирование (ООП) – Это парадигма программирования, основанная на концепции “объектов”, которые могут содержать данные и код: данные в виде полей (часто известных как атрибуты или свойства), а код - в виде процедур (часто известных как методы).

Функциональное программирование – Это парадигма программирования, которая обрабатывает вычисления как оценку математических функций и избегает изменяемого состояния и изменяемых данных.

Логическое программирование – Это парадигма программирования, в которой программа представляет собой набор деклараций (“фактов” и “правил”) в формальной логике. Вычисления выполняются путём применения правил к фактам для выведения новых фактов.

* 1. Стандарт оформления программного кода – это набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Стандарты оформления кода могут включать в себя правила именования переменных, отступов, комментариев, расположения скобок и другие элементы.

Качественное формирование листинга программы важно по нескольким причинам. Во-первых, хорошо оформленный код легче читать и понимать, что упрощает процесс разработки и отладки программы. Во-вторых, соблюдение стандартов оформления кода облегчает работу в команде, так как все разработчики используют единый стиль написания кода.

Правила формирования листинга программы:

* + 1. Используйте понятные имена переменных, которые отражают их назначение.
    2. Следуйте соглашениям об именовании переменных, функций и других элементов программы.
    3. Используйте отступы для визуального выделения блоков кода.
    4. Размещайте открывающие и закрывающие скобки на отдельных строках или на одной строке в зависимости от принятого стиля оформления кода.
    5. Оставляйте комментарии к коду, объясняющие его работу.
    6. Структурируйте код программы, разбивая его на функции и модули.
    7. Венгерская нотация: Это соглашение об именовании переменных, при котором имя переменной начинается с префикса, указывающего на тип данных переменной. Например, переменная целого типа может иметь имя iCount, где префикс i указывает на то, что это переменная целого типа.
    8. Верблюжья нотация: Это соглашение об именовании, при котором несколько слов в имени переменной или функции записываются слитно, а каждое последующее слово начинается с заглавной буквы. Например, camelCase или CamelCase.
    9. Грамотный выбор идентификаторов: При выборе имён для переменных, констант, структур и других элементов программы важно следовать определенным правилам и соглашениям. Имя должно быть понятным и отражать назначение элемента. Также важно соблюдать соглашения о длине имени идентификатора и использовать только допустимые символы.
    10. Отступы в программном коде: Отступы используются для визуального выделения блоков кода. Горизонтальные отступы обычно используются для выделения вложенных блоков кода, а вертикальные - для разделения логически связанных строк кода. Отступы могут быть выполнены с помощью пробелов или символов табуляции.
    11. Использование комментариев: Комментарии используются для пояснения кода программы. Они позволяют программисту оставить заметки для себя или других разработчиков, объясняющие, как работает тот или иной фрагмент кода.
    12. Разбиение задачи на подзадачи: Разбиение задачи на подзадачи позволяет упростить процесс разработки программы. Каждая подзадача может быть решена отдельно, а затем результаты могут быть объединены для решения общей задачи.
    13. Правила оформления процедур, функций, модулей, классов: При оформлении процедур, функций, модулей и классов важно следовать определенным правилам и соглашениям. Например, каждая процедура или функция должна иметь понятное имя и содержать комментарии, объясняющие её назначение и работу. Модули и классы должны быть организованы логически и содержать только те элементы, которые относятся к их назначению.