КОМИТЕТ ОБРАЗОВАНИЯ, НАУКИ И МОЛОДЕЖНОЙ ПОЛИТИКИ ВОЛГОГРАДСКОЙ ОБЛАСТИ
государственное бюджетное профессиональное образовательное учреждение
«Волгоградский колледж управления и новых технологий имени Юрия Гагарина»
(ГБПОУ «ВКУиНТ им. Ю. Гагарина»)

УЧЕБНОЕ ПОСОБИЕ

по учебной дисциплине
Основы алгоритмизации и программирование

РАЗДЕЛ 1. Основные принципы алгоритмизации и программирования

Для студентов с ОВЗ (нарушение слуха),
обучающихся по специальности
09.02.07 Информационные системы и программирование

Волгоград
2023

УТВЕРЖДАЮ
Зам. директора по УиМР
_____ Солодова Т.Е.
«26» 10 2023г.

Разработчик: преподаватель ГБПОУ «ВКУиНТ им.Ю. Гагарина»

Гладкова Е.М. _____

РАССМОТРЕНО И ОДОБРЕНО
Кафедрой Цифровых технологий в промышленности

Протокол № 2 от «24» 10 2023г.
Заведующий кафедрой _____
подпись

РЕКОМЕНДОВАНО
Научно-методическим советом
ГБПОУ «ВКУиНТ им. Ю. Гагарина»
Протокол № 2 от «26» 10 2023г.

СОДЕРЖАНИЕ

Пояснительная записка

Адаптированное учебное пособие по учебной дисциплине «Основы алгоритмизации и программирования» разработано для студентов, обучающихся по специальности 09.02.07 Информационные системы и программирование для лиц о ограниченными возможностями здоровья (нарушение слуха).

Цели и планируемые результаты освоения дисциплины

В результате освоения дисциплины обучающийся должен ***уметь***:

- Разрабатывать алгоритмы для конкретных задач.
- Использовать программы для графического отображения алгоритмов.
- Определять сложность работы алгоритмов.
- Работать в среде программирования.
- Реализовывать построенные алгоритмы в виде программ на конкретном языке программирования.
- Оформлять код программы в соответствии со стандартом кодирования.
- Выполнять проверку, отладку кода программы.

В результате освоения дисциплины обучающийся должен ***знать***:

- Понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции.
- Эволюцию языков программирования, их классификацию, понятие системы программирования.
- Основные элементы языка, структуру программы, операторы и операции, управляющие структуры, структуры данных, файлы, классы памяти.
- Подпрограммы, составление библиотек подпрограмм.
- Объектно-ориентированную модель программирования, основные принципы объектно-ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляции и полиморфизма, наследования.

В результате освоения дисциплины обучающийся осваивает элементы компетенций:

	Наименование компетенций
ОК 1	Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам
ОК 2	Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности
ОК 4	Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами

ОК 5	Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста
ОК 9	Использовать информационные технологии в профессиональной деятельности
ОК 10	Пользоваться профессиональной документацией на государственном иностранном языках
ПК 1.1	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.
ПК 1.2	Разрабатывать программные модули в соответствии с техническим заданием.
ПК 1.3	Выполнять отладку программных модулей с использованием специализированных программных средств.
ПК 1.4	Выполнять тестирование программных модулей.
ПК 1.5	Осуществлять рефакторинг и оптимизацию программного кода.
ПК 1.6	Разрабатывать модули программного обеспечения для мобильных платформ.
ПК 2.4	Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.
ПК 2.5	Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Объем учебного материала:

Вид учебной работы	**Объем часов**
Максимальная учебная нагрузка	**150**
Самостоятельная работа	**20**
Объем учебной нагрузки во взаимодействии с преподавателем	**114**
в том числе:	
теоретическое обучение	58
лабораторные работы	56
практические занятия	—
курсовая работа/проект *(если предусмотрено)*	—
контрольная работа	—
консультации	8
Промежуточная аттестация в форме	экзамена

Материал учебного пособия охватывает ***первый раздел «Основные принципы алгоритмизации и программирования учебной дисциплины»***, рассчитанный на **16 часов**. Раздел состоит из восьми лекций. Каждая лекция

содержит большое количество рисунков, схем, таблиц для лучшего изучения материала по курсу. В конце каждой лекции приводятся контрольные вопросы. Материал пособия завершается комплектом заданий для закрепления материала.

Методические рекомендации по работе с обучающимися данной нозологии

Слабослышащие люди - это люди с частичной слуховой недостаточностью, что затрудняет речевое развитие, но сохраняется возможность самостоятельного накопления речевого запаса. К слабослышащим относятся люди с очень большими различиями в области слухового восприятия: от небольшого нарушения восприятии шепотной речи до резкого ограничения восприятия речи разговорной громкости.

Выделены две категории слабослышащих:

1)слабослышащие, которые имеют тяжёлое недоразвитие речи; они говорят

Отдельные слова, короткие фразы, может быть неправильное построение фразы;

2) слабослышащие, которые владеют развёрнутой фразовой речью с небольшими отклонениями в грамматическом строе, фонетическом оформлении.

Специфика обучения лиц с нарушением слуха:

- на занятии студента с нарушением слуха следует посадить за первый стол (первую парту);
- при объяснении материала преподаватель должен стоять лицом к студенту;
- осуществлять опору на сохранные анализаторы: зрительный, обонятельный, тактильный, вибрационный;
- использовать письменную речь, дублирование звуковой информации визуальной;
- речь должна быть эмоционально выразительной, с четкой артикуляцией, следует говорить громче и четче, подбирая подходящий уровень;
- использование информационных технологий;
- обеспечение звуковыми средствами воспроизведения информации;
- использованием жестового языка (сурдоперевода, тифлосурдоперевода);
- использование дактильной речи,
- сочетание жестового языка со всеми видами речевой деятельности: говорение, слушание, чтение, письмо, дактилирование, зрительное восприятие с лица и с руки говорящего);

- применение опережающего чтения, когда студенты заранее знакомятся с лекционным материалом и обращают внимание на незнакомые и непонятные слова и фрагменты;
- дополнительное объяснение некоторых основных понятий изучаемого
- материала с использованием наглядного материала, видеоматериалов;
- использование различных образовательных технологий, в том числе
- дистанционных, электронного обучения;
- при необходимости повторно объяснять материал;
- соблюдение слухоречевого режима на занятии.

В процессе обучения рекомендуется использовать:

- опорные конспекты, схемы, которые придают упрощенный схематический вид изучаемым понятиям;
- видеоинформацию, которая сопровождается текстовой бегущей строкой или сурдологическим переводом;
- анимацию с гиперссылками для изображения различных динамических
- моделей, не поддающихся видеозаписи.

Чтобы обучающиеся с нарушенным слухом получали информацию в полном объеме, звуковую информацию нужно обязательно дублировать зрительной. Особую роль играют видеоматериалы. Видеоинформация может сопровождаться текстовой бегущей строкой или сурдологическим перевод ом. Видеоматериалы особенно помогают в изучении процессов и явлений, поддающихся видеозаписи, а анимация может быть использована для изображения различных динамических моделей, не поддающихся видеозаписи процессов и явлений. Анимация может сопровождаться гиперссылкам и, которые комментируют отдельные элементы. Важную обучающую функцию выполняют компьютерные модели, конструкторы, компьютерный лабораторный практикум.

Раздел 1. Основные принципы алгоритмизации и программирования

Тема 1.1 Понятие и свойства алгоритма. Способы описания алгоритмов

План занятия

1. Понятие алгоритма
2. Свойства алгоритма
3. Способы описания алгоритмов
4. Примеры решения задач

1. Понятие алгоритма

Для решения любой задачи последовательность действий может быть следующей

- Выбрать способ(метод) решения задачи и изучить его во всех подробностях;
- Сообщить исполнителю выбранный метод в абсолютно понятном для него виде.

Первый этап этого процесса обычно не вызывает затруднений. Главная трудность этого этапа – выбрать из нескольких методов более подходящий для решения данной задачи: наименее трудоемкий, максимально эффективный и т.д.

Второй этап значительно сложнее. Дело в том, что если способ (метод) решения задачи описан произвольно, то нет гарантии, что он будет верно понят исполнителем.

Описание метода решения задачи следует выполнять в соответствии с определенными правилами, а именно:

- выделить величины, являющиеся исходными для задачи;
- разбить процесс решения задачи на такие этапы, которые известны исполнителю и которые он может выполнить однозначно без всяких пояснений;
- указать порядок выполнения этапов;
- указать признак окончания процесса решения задачи;
- указать во всех случаях, что является результатом решения задачи.

Описание метода, выполненное в соответствии с этими правилами, называется ***алгоритмом решения задачи.***

Алгоритм *– это метод (способ) решения задачи, записанный по определенным правилам, обеспечивающим однозначность его понимания и механического исполнения при всех значениях исходных данных (из некоторого множества значений).*

Существует и более короткое определение: ***алгоритм*** - *это строго определенная последовательность действий, необходимых для решения данной задачи.*

2. Свойства алгоритма

Алгоритм обладает определенными свойствами.

1. ***Дискретность алгоритма***. Это свойство означает, что решение задачи, записанное в виде алгоритма, разбито на отдельные простейшие команды, которые расположены в порядке их выполнения.
2. ***Определенность алгоритма***. Это свойство означает, что каждая команда алгоритма должна быть понятна исполнителю, не оставлять места для ее неоднозначного толкования и неопределенного исполнения.
3. ***Результативность алгоритма***. Свойство алгоритма, состоящее в том, что он всегда приводит к результату через конечное число шагов.
4. **Массовость алгоритма**. Это свойство заключается в том, что каждый алгоритм, разработанный для решения некоторой задачи, должен быть применим для решения задач этого типа при всех допустимых значениях исходных данных.

Рецепт блюда, инструкция по сборке мебели, компьютерная программа — все это примеры алгоритмов.

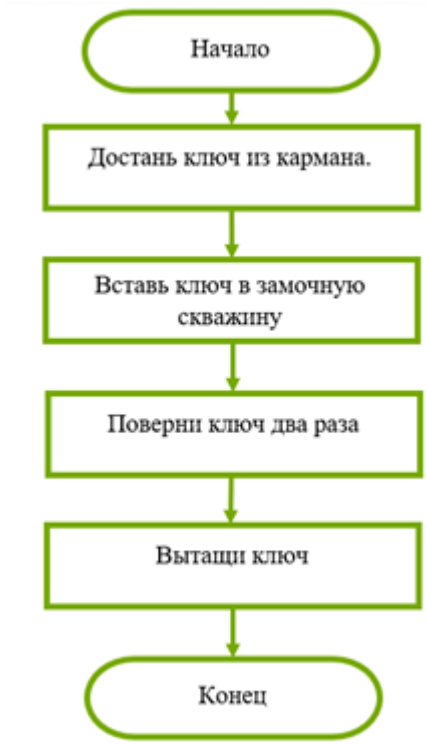


Рисунок 1. Алгоритм открытия замка

3. Способы описания алгоритмов

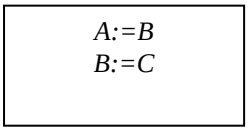
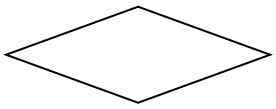
Существует несколько способов описания алгоритмов.

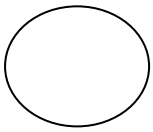
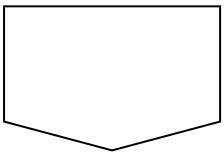
1. *Словесно-формульное* описание алгоритма, т.е. описание алгоритма с помощью слов и формул.
2. *Графическое* описание алгоритма, т.е. описание с помощью специальных графических схем алгоритмов – ***блок-схем***.

Блок-схема алгоритма представляет собой систему связанных геометрических фигур.

Каждая фигура обозначает один этап решения задачи и называется ***блоком***. Порядок выполнения этапов указывается стрелками, соединяющими блоки. В схеме блоки стараются размещать сверху вниз в порядке их выполнения. Для наглядности операции разного вида изображаются в схеме различными геометрическими фигурами.

Таблица 1. Блоки для построения алгоритмов

№	Блок	Наименование	Содержание
1.		Блок вычислений	Вычислительные действия или последовательность действий
2.		Логический блок	Выбор выполнения алгоритма в зависимости от некоторого условия
3.		Блоки ввода-вывода данных	1. Общие обозначения в/вывода данных 2. Вывод данных, носителем которых является документ
4.		Начало (конец)	Начало или конец алгоритма, вход или выход а подпрограмму
5.		Процесс пользователя (подпрограмма)	Вычисление по стандартной программе или подпрограмме

6.		Блок модификации	Функция выполняет действия, изменяющие пункты алгоритма (заголовок цикла)
7.		Соединитель	Указание связи прерванными линиями между потоками информации в пределах одного листа
8.		Межстраничные соединения	Указание связи между информацией на разных листах

4. Примеры решения задач

Пример 1. Составить алгоритм начисления зарплаты с помощью словесно-формульного описания согласно следующему правилу: «Если стаж работы сотрудника менее 5 лет, то зарплата 35 000 руб., при стаже работы от 5 до 15 лет 50 000 руб., при стаже свыше 15 лет зарплата повышается с каждым годом на 1 000 рублей».

Формулировка задачи в математическом виде:

Вычислить

35 000, если ST<5

ZP= { 50 000, если 5<=ST<=15

35 000 + (ST-15)* 1 000, если 15<ST

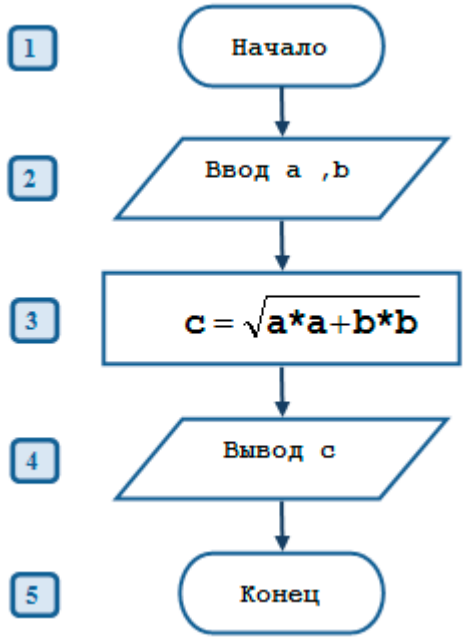
где ZP – зарплата; ST – стаж работы.

Словесно-формульное описание алгоритма решения задачи:

1. Ввести ST, перейти к п.2.
2. Если ST<5, то ZP:=35 000, перейти к п.4, иначе – перейти к п.3.
3. Если 5<=ST<=15, то ZP:=50 000, перейти к п.4, иначе ZP:=50 000+(ST-15)*1 000, перейти к п.4.
4. Вывести значение ZP, перейти к п.5.
5. Конец работы.

Пример 2. Блок-схема алгоритма для вычисления гипотенузы по двум

известным значениям катетов



Контрольные вопросы

1. Что такое алгоритм.
2. Перечислить свойства алгоритма.
3. Какие существуют способы описания алгоритмов?
4. Какие блоки можно использовать при построении алгоритмов?
5. В каких случаях следует использовать логический блок, блок модификации?

Тема 2. Базовые структуры алгоритмов

План занятия

1. Алгоритм линейной структуры.
2. Алгоритм разветвляющейся структуры
3. Алгоритм циклической структуры

1. Алгоритм линейной структуры

Базовые структуры алгоритмов – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий. К основным структурам относятся следующие: линейные, разветвляющиеся, циклические. Линейный алгоритм – алгоритм, в котором действия осуществляются последовательно.

Структура линейного алгоритма имеет следующий вид:

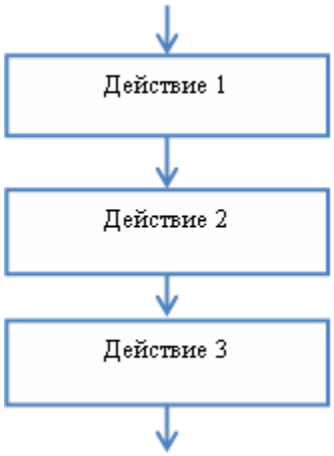


Рисунок 2. Структура линейного алгоритма

Любой человек ежедневно встречается с множеством задач: от самых простых и хорошо известных до очень сложных. Для многих задач существуют определенные правила (инструкции, предписания), объясняющие исполнителю, как решать данную задачу. Эти правила человек может изучить заранее или сформулировать сам в процессе решения задачи. Чем точнее и понятнее будут описаны правила решения задач, тем быстрее человек овладеет ими и будет эффективнее их применять.

Каждый человек ежедневно использует различные алгоритмы: инструкции, правила, рецепты и т. п. Обычно мы это делаем не задумываясь. Например, открывая дверь ключом, никто не размышляет над тем, в какой последовательности будет выполнять действия. Однако, чтобы кого – нибудь научить открывать дверь, придется четко указать и сами действия, и порядок их выполнения. Например, так:

- Достать ключ.

- Вставить ключ в замочную скважину.
- Повернуть ключ 2 раза против часовой стрелки.
- Вынуть ключ.

Таким образом, структура линейного алгоритма будет напоминать подробную инструкцию по выполнению определенных действий.

Рисунок 3. Алгоритм приготовления кофе

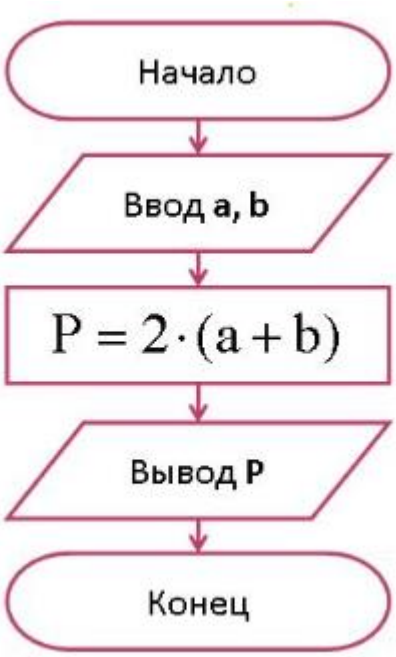


Рисунок 4. Алгоритм нахождения периметра прямоугольника

2. Алгоритм разветвляющейся структуры

Разветвляющийся – алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи, в зависимости от выполнения условий.

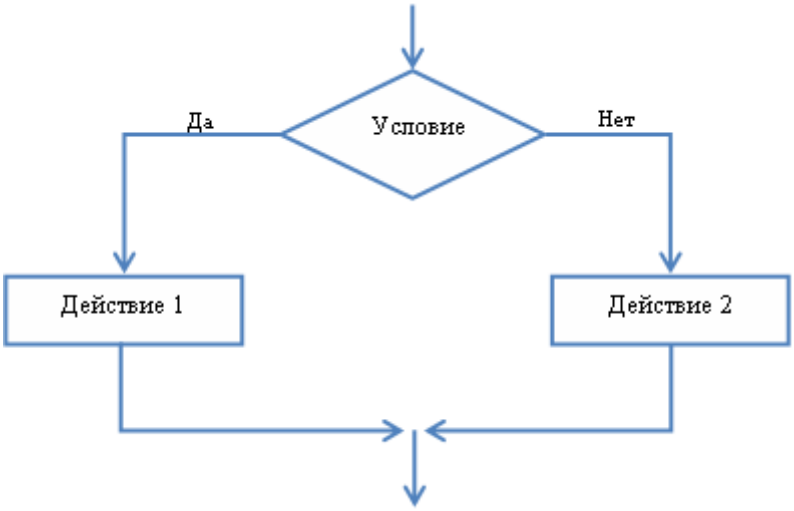


Рисунок 5. Структура ветвления

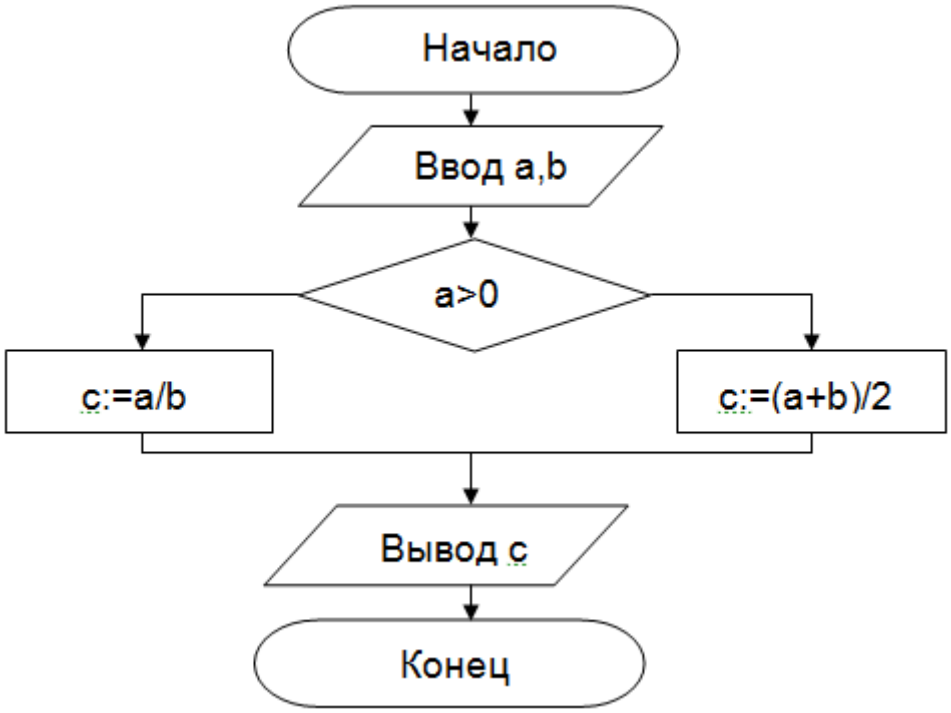


Рисунок 6. Пример алгоритма определения среднего арифметическое двух чисел, если а положительное и частное (a/b) в противном случае.

3. Алгоритм циклической структуры

Циклический – алгоритм, в котором тело цикла (последовательность команд) выполняется многократно. **Циклом** называется последовательность действий, выполняемых многократно, каждый раз при новых значениях параметров.

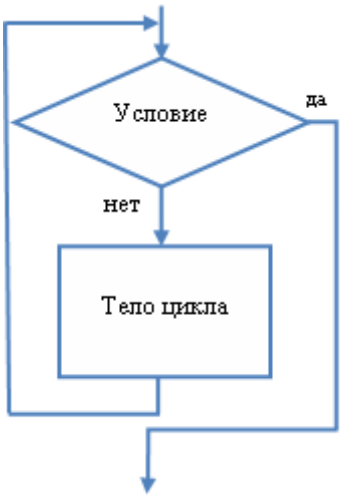


Рисунок 7. Цикл с предусловием

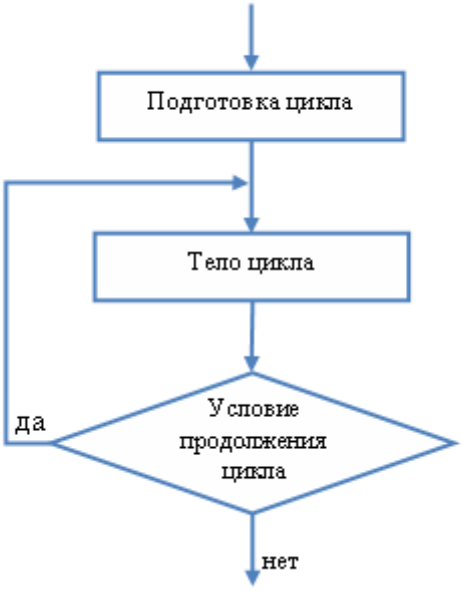


Рисунок 8 . Цикл с постусловием

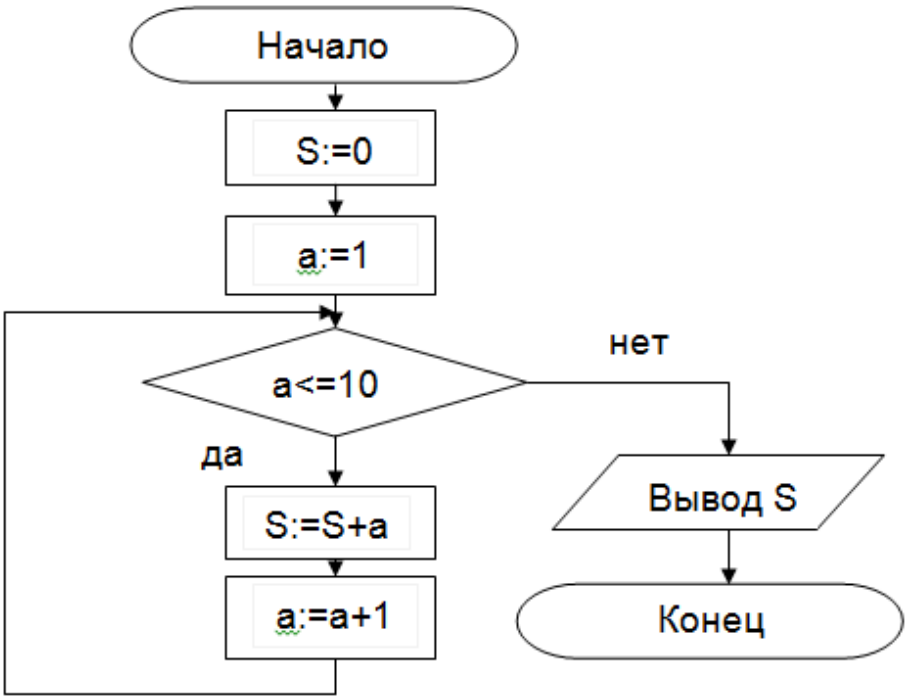


Рисунок 9. Алгоритм нахождения суммы целых чисел в диапазоне от 1 до 10

(алгоритм с предусловием)

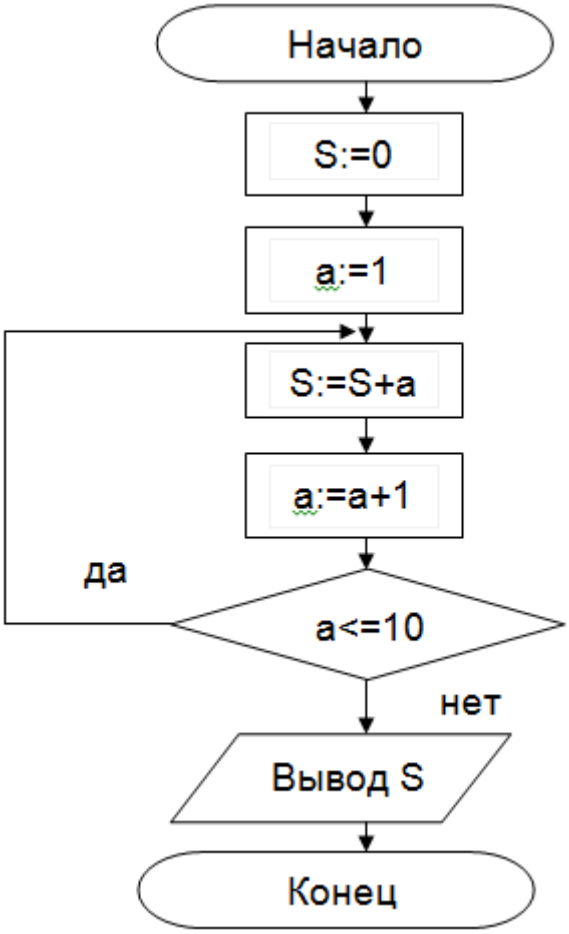


Рисунок 10. Алгоритм нахождения суммы целых чисел в диапазоне от 1 до 10 (алгоритм с постусловием)

Контрольные вопросы

1. Что представляют собой базовые структуры алгоритмов
2. Какой алгоритм называется алгоритмом линейной структуры
3. Что такое разветвляющийся алгоритм
4. Какой алгоритм называется циклическим? Какие блоки используются для его построения?

Тема 3. Языки программирования: эволюция, поколения

План занятия

1. Язык программирования: определение, поколения
2. Классификация языков программирования
3. Перспективы развития языков программирования

1. Язык программирования: определение, поколения

Язык программирования представляет собой совокупность символов и правил их использования для описания процессов решения задач на ЭВМ.

В развитии инструментального программного обеспечения (т.е. программного обеспечения, служащего для создания программных средств в любой программной области) рассматривают пять поколений языков программирования (ЯП).

Языки программирования как средство общения человека с ЭВМ от поколения к поколению улучшали свои характеристики, становясь все более доступными в освоении непрофессионалам.

Поколения языков программирования

Первое поколение

Языки программирования *первого поколения* представляли собой набор машинных команд в двоичном или восьмеричном формате, который определялся архитектурой конкретной ЭВМ. Каждый тип ЭВМ имел свой язык программирования, программы на котором были пригодны только для данного типа ЭВМ. От программиста при этом требовалось хорошее знание не только машинного языка, но и архитектуры ЭВМ.

Машинный язык представляет собой систему команд конкретной ЭВМ и реализуется ею непосредственно. При разработке программы на машинном языке необходимо прежде всего распределить память под переменные и константы, т.е. для каждой переменной или константы выделить несколько ячеек памяти. Для задания, например, константы в выделенную область памяти записывается соответствующее значение.

Программа на машинном языке представляет собой последовательность машинных команд и фиксированных областей памяти, выделенных под переменные и константы. Структура машинной программы не фиксирована, так как переменные и константы могут чередоваться с командами в любом порядке. При этом никакой разницы между элементами памяти, в которых содержатся команды, константы или переменные, нет. Так, элемент памяти, отведенный под команду, может быть использован как переменная или константа. Контроль за правильностью их использования осуществляет только программист. Это приводит к большому числу ошибок, которые иногда очень трудно обнаружить. Действия, выполняемые машинными командами, элементарны: например, переслать содержимое одной ячейки памяти в

другую, сложить содержимое двух ячеек и т.д. Между тем запись программы с помощью таких действий требует больших трудозатрат. Машинный язык позволяет использовать все возможности аппаратуры ЭВМ. С его помощью можно создавать достаточно эффективные программы.

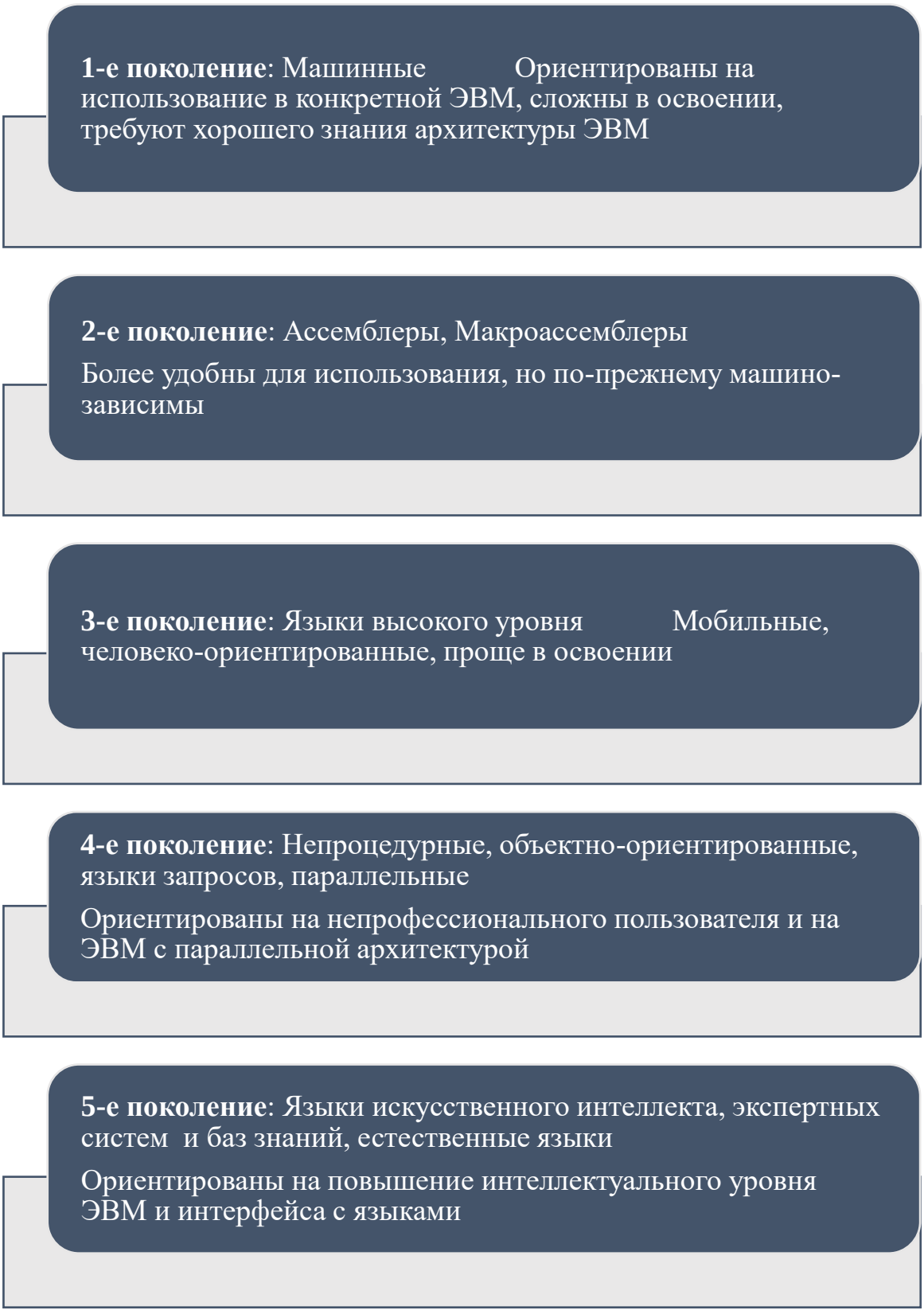


Рисунок 11. Поколения языков программирования

Второе поколение языков программирования характеризуется созданием языков ассемблерного типа (ассемблеров, макроассемблеров), позволяющих вместо двоичных и других форматов машинных команд использовать их символьные обозначения (имена). Являясь существенным шагом вперед, ассемблерные языки все еще оставались машинно-зависимыми, а программист все также должен был быть хорошо знаком с организацией и функционированием аппаратной среды конкретного типа ЭВМ. При этом ассемблерные программы все также затруднительны для чтения, трудоемки при отладке и требуют больших усилий для переноса на другие типы ЭВМ.

Языки уровня Ассемблера являются машинно-ориентированными, соответствующими системам команд конкретных ЭВМ, но позволяющими составлять программы в форме, более удобной для человека. Преимуществом языка Ассемблера является символическая адресация, когда командам, константам и переменным присваиваются некоторые имена, по которым к ним можно обращаться. Предусматриваются также средства соединения нескольких программ в единый программный модуль и средства контроля ошибок.

На языках Ассемблера пишутся эффективные программы, позволяющие использовать все возможности ЭВМ.

К недостаткам относятся излишняя детализация записи программ, отсутствие контроля за обращением к элементам памяти.

Третье поколение

Это поколение начинается с появления в 1956 г. первого языка высокого уровня - Fortran. Вскоре после языка Fortran появились такие языки, как Algol, Cobo1, Basic, PL/1, Pascal, APL, ADA, C, Forth, Lisp, Modula и др.

Языки высокого уровня делятся на проблемно-ориентированные и процедурно-ориентированные.

Проблемно-ориентированные языки предназначены для решения узкого класса задач. При программировании производится описание задачи, а не алгоритм ее решения.

Процедурно-ориентированные языки созданы для описания алгоритмов решения задач. При этом различают машинно-зависимые и машинно-независимые алгоритмические языки.

Машинно-зависимые языки высокого уровня дают возможность полностью использовать аппаратуру ЭВМ, написать ясные и удобочитаемые программы. Они в меньшей степени машинно-зависимы, чем языки Ассемблера, так как операторы и выражения записываются в форме, более удобной для человека, например язык ПЛ/М. Однако они во многом зависят от реализации конкретной машины, и этот недостаток не позволяет им получить широкого распространения.

Машинно-независимые языки высокого уровня, или просто алгоритмические языки, не содержат машинно-зависимых операторов. К языкам этого типа можно отнести Фортран, Алгол, Бейсик, Фокал, ПЛ/1,

Паскаль и др. Основными достоинствами программирования на этих языках являются высокая производительность труда программистов, простота эксплуатации программ, возможность их переноса с одной машины на другую, наличие средств контроля. Специальная программа-транслятор, написанная на внутреннем языке машины и заранее введенная в память ЭВМ, обрабатывает символическое описание алгоритма, представленное на алгоритмическом языке, и осуществляет автоматический перевод программы на внутренний язык машины.

Четвертое поколение

Языки *четвертого поколения* носят ярко выраженный *непроцедурный* характер, определяемый тем, что программы на таких языках описывают только *что,* а не *как* надо сделать. В программах формируются скорее соотношения, а не последовательности шагов выполнения алгоритмов. Типичными примерами непроцедурных языков являются языки. используемые для задач искусственного интеллекта (Prolog, Langin). Так как непроцедурные языки имеют минимальное число синтаксических правил, они значительно более пригодны для применения непрофессионалами в области программирования.

Второй тенденцией развития языков программирования четвертого поколения являются *объектно-ориентированные языки*, базирующиеся на понятии программного объекта, впервые использованного в языке Simula-67 и составившего впоследствии основу языка SmallTa1k. Примеры объектно-ориентированных языков: Pascal, Basic, C++, SmallTalk, Simula, Actor .

Третьим направлением языков четвертого поколения можно считать языки запросов, позволяющих пользователю получать информацию из баз данных. Среди языков запросов фактическим стандартом стал язык SQL.

Четвертым направлением развития являются языки параллельного программирования (модификация ЯВУ Fortran, языки Occam, SISAL, FP и др.), которые ориентированы на создание программного обеспечения для вычислительных средств параллельной архитектуры (многомашинные, мультипроцессорные среды и др.), в отличие от языков третьего поколения, ориентированных на традиционную однопроцессорную архитектуру.

Пятое поколение

К интенсивно развивающемуся в настоящее время *пятому поколению* относятся языки искусственного интеллекта, экспертных систем, баз знаний (InterLisp, ExpertLisp, IQLisp, SALL и др.), а также естественные языки, не требующие освоения какого-либо специального синтаксиса (в настоящее время успешно используются естественные ЯП с ограниченными возможностями – Clout, Q&A, HAL и др.).

2. Классификация языков программирования

Существует огромное количество языков программирования. Их все можно классифицировать по следующим факторам: уровню языка, его

специализации, и также алгоритмичности.

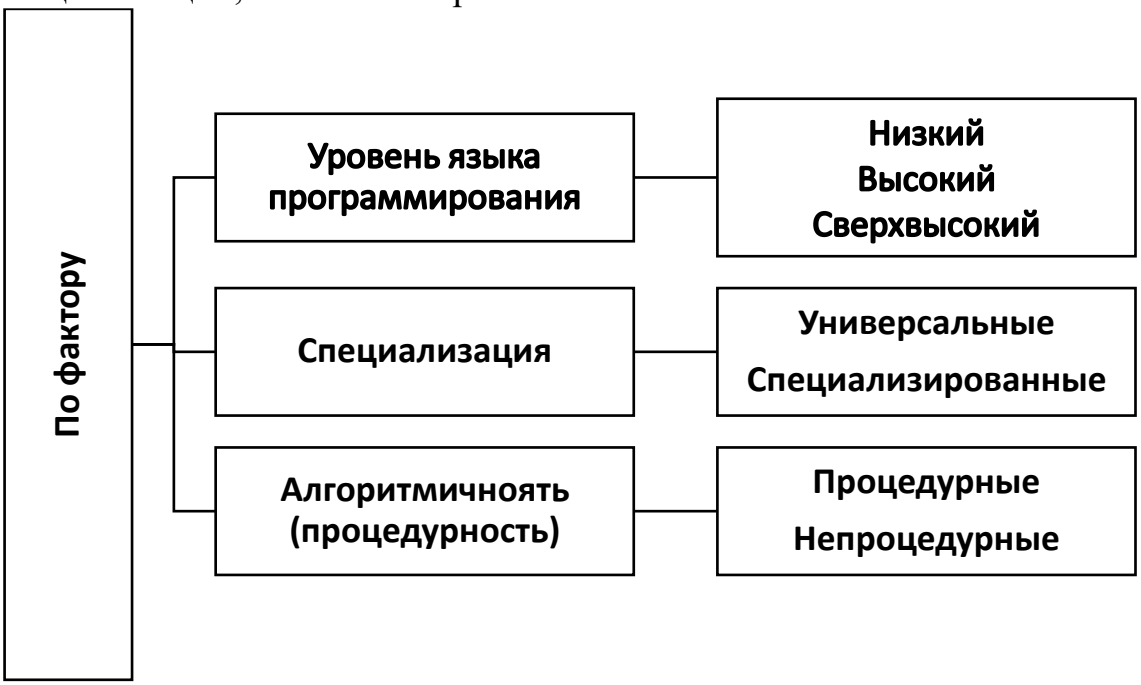


Рисунок 12. Классификация языков программирования по трем факторам

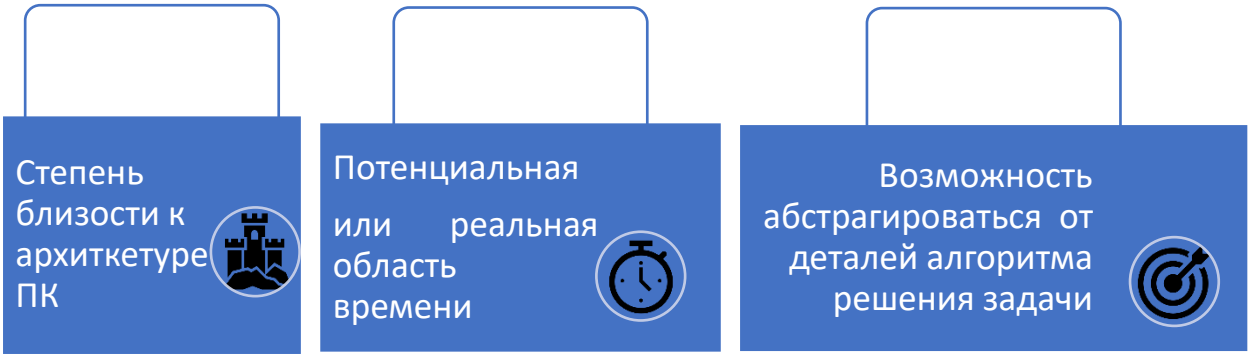


Рисунок 13. Классификация по области применения

1. Машинно-зависимые языки
2. Язык ассемблера, Макроассемблеры, Fortran
3. Algol, Cobo1, Basic, PL/1, Pascal, APL, ADA, C, Forth, Lisp,
4. C++, C#, Delphi, VBA, JavaScript, Python, Ruby, Perl, PHP
5. InterLisp, ExpertLisp, IQLisp, SALL, Clout, Q&A, HAL

Рисунок 14. Примеры языков программирования

3 Перспективы развития языков программирования

Язык программирования является тем незаменимым инструментом, который служит программисту для создания программного обеспечения. Чем лучше язык программирования, тем более совершенную программу удается написать.

Создание ПО во многом можно сравнить с производством, где среди прочих важнейших факторов, определяющими являются: производительность труда команды разработчиков, издержки и качество конечного продукта. Все разрабатываемые технологии создания программ должны поддерживаться языками программирования.

С каждым днем задачи, решаемые с использованием компьютеров, становятся все сложнее и разнообразней. Это приводит к тому, что лучшие умы планеты в лице талантливых разработчиков, стремятся к созданию новых, более мощных, ориентированных на проблемную область, языков программирования.

Языки программирования должны помогать разработчикам в нелегкой борьбе за продление жизненного цикла программ. Ведь не актуальный, морально и технически устаревший продукт мало кому интересен.

Можно предположить, что дальнейший процесс разработки новых языков программирования в ближайшее время будет двигаться в направлении все большей абстракции. Основные программы программирования будут стремиться к изменению уровня детализации, наибольшему упрощению. Это приведет к повышению надежности процесса создания ПО как такового и уменьшению количества допускаемых разработчиками ошибок.

Контрольные вопросы

1. Что такое язык программирования.
2. Перечислите и опишите поколения языков программирования.
3. Приведите примеры языков программирования для каждого поколения.
4. По каким факторам можно классифицировать языки программирования?

4. Системы программирования: назначение, классификация

План занятия

1. Система программирования: основные характеристики
2. Классификация систем программирования

1. Система программирования: основные характеристики

Система программирования представляет собой совокупность средств разработки программ (языки программирования, текстовые редакторы, трансляторы, редакторы связей, библиотеки подпрограмм, утилиты и обслуживающие программы), обеспечивающих автоматизацию составления и отладки программ пользователя. Система программирования - программная система, предназначенная для разработки программ на конкретном языке программирования.

Система программирования включает следующие программные компоненты:

- редактор текста;
- транслятор с соответствующего языка;
- компоновщик (редактор связей);
- отладчик;
- библиотеки подпрограмм.

Редактор текста - это программа для ввода и модификации текста.

Трансляторы предназначены для преобразования программ, написанных на языках программирования, в программы на машинном языке. Программа, подготовленная на каком-либо языке программирования, называется исходным модулем. В качестве входной информации трансляторы применяют исходные модули и формируют в результате своей работы объектные модули, являющиеся входной информацией для редактора связей. Объектный модуль содержит текст программы на машинном языке и дополнительную информацию, обеспечивающую настройку модуля по месту его загрузки и объединение этого модуля с другими независимо оттранслированными модулями в единую программу.

Трансляторы делятся на два класса: компиляторы и интерпретаторы. Компиляторы переводят весь исходный модуль на машинный язык. Интерпретатор последовательно переводит на машинный язык и выполнят операторы исходного модуля

Преимущество интерпретатора перед компилятором состоит в том, что программа пользователя имеет одно представление - в виде текста. При компиляции одна и та же программа имеет несколько представлений - в виде текста и в виде выполняемого файла.

Компоновщик, или редактор связей - системная обрабатывающая программа, редактирующая и объединяющая объектные (ранее оттраслированные) модули в единые загрузочные, готовые к выполнению программные модули. Загрузочный модуль может быть помещен ОС в основную память и выполнен.

Отладчик позволяет управлять процессом исполнения программы, является инструментом для поиска и исправления ошибок в программе. Базовый набор функций отладчика включает:

- пошаговое выполнение программы (режим трассировки) с отображением результатов,
- остановка в заранее определенных точках,
- возможность остановки в некотором месте программы при выполнении некоторого условия;
- изображение и изменение значений переменных.

Загрузчик - системная обрабатывающая программа. Загрузчик помещает объектные и загрузочные модули в оперативную память, объединяет их в единую программу, корректирует перемещаемые адресные константы с учетом фактического адреса загрузки и передает управление в точку входа созданной программы.

2. Классификация систем программирования

Системы программирования классифицируют по признакам, приведенным на рисунке:

Рисунок 15. Классификация систем программирования

Следует отметить, что:

- Отличительной особенностью многоязыковых систем является то, что отдельные части (секции, модули или сегменты) программы

могут быть подготовлены на различных языках и объединены во время или перед выполнением в единый модуль;

- В открытую систему можно ввести новый входной язык с транслятором, не требуя изменений в системе;
- В интерпретирующей системе осуществляется покомандная расшифровка и выполнение инструкций входного языка (в среде данной системы программирования); в компилирующей – подготовка результирующего модуля, который может выполняться на ЭВМ практически независимо от среды.

Контрольные вопросы

1. Что такое система программирования.
2. Какие функции выполняет транслятор?
3. Какой программный компонент системы программирования позволяет управлять процессом исполнения программы и является инструментом для поиска и исправления ошибок в программе?
4. Назначение загрузчика.
5. Приведите примеры современных систем программирования.

Тема 5. Процесс разработки программы в системе программирования

План занятия

1. Этапы разработки программы в системе программирования
2. Жизненный цикл программного обеспечения

1. Этапы разработки программы в системе программирования

Процесс разработки программного обеспечения (англ. software development process) — процесс, посредством которого потребности пользователей преобразуются в программный продукт. Процесс разработки программного обеспечения является составной частью программной инженерии.

Ввод. Программа на исходном языке (исходный модуль) готовится с помощью текстовых редакторов и в виде текстового файла или раздела

библиотеки поступает на вход транслятора.

Трансляция. Трансляция исходной программы есть процедура преобразования исходного модуля в промежуточную, так называемую объектную форму. Трансляция в общем случае включает в себя препроцессинг (предобработку) и компиляцию.

Препроцессинг – необязательная фаза, состоящая в анализе исходного текста, извлечения из него директив препроцессора и их выполнения. Директивы препроцессора представляют собой помеченные символами (обычно %, #, &) строки, содержащие аббревиатуры или другие символические обозначения конструкций, включаемых в состав исходной программы перед ее обработкой компилятором. Данные для расширения исходного текста могут быть стандартными, определяться пользователем либо содержаться в системных библиотеках ОС.

Компиляция – в общем случае многоступенчатый процесс, включающий следующие фазы:

- *Синтаксический анализ* – проверка правильности конструкций, использованных программистом при подготовке текста;
- *Семантический анализ* – выявление несоответствий типов и структур переменных, функций и процедур;
- *Генерация объектного кода* – завершающая фаза трансляции.

Объектный модуль – текст программы на машинном языке, включающий машинные инструкции, словари, служебную информацию.

Построение исполнительного модуля. Построение загрузочного модуля осуществляется специальными программными средствами – редактором связей, построителем задач, компоновщиком, основной функцией которых является объединение объектных и загрузочных модулей в единый загрузочный модуль с последующей записью в библиотеку или файл. Полученный модуль в дальнейшем может использоваться для сборки других программ и т.д., что создает возможность наращивания программного обеспечения.

Загрузка программы. Загрузочный модуль после сборки либо помещается в качестве раздела в пользовательскую библиотеку программ, либо в качестве последовательного файла на накопителе на магнитном диске (НМД). Выполнение модуля состоит в загрузке его в оперативную память, настройке по месту в памяти и передаче ему управления. *Образ загрузочного модуля в памяти называется абсолютным модулем, поскольку все команды ЭВМ здесь приобретают окончательную форму и получают абсолютные адреса в памяти.*

Современные системы программирования позволяют удобно переходить от одного этапа к другому. Это осуществляется в рамках так называемой среды интегрированной среды программирования, которая содержит в себе текстовый редактор, компилятор, компоновщик, встроенный отладчик и, в зависимости от системы или ее версии, предоставляет программисту дополнительные удобства для написания и отладки программ.

<table>
<tr><td rowspan="3">  </td><td>Исходный текст</td><td rowspan="3">Ввод</td></tr>
<tr><td>Текстовый редактор</td></tr>
<tr><td>Исходный модуль</td></tr>
<tr><td rowspan="2">Библиотека системных программ</td><td>Препроцессор</td><td rowspan="2">Препроцессинг</td></tr>
<tr><td>Расширенный модуль</td></tr>
<tr><td rowspan="2">Библиотека программ пользователя</td><td>Транслятор (компилятор)</td><td rowspan="2">Трансляция (компиляция)</td></tr>
<tr><td>Объектный модуль</td></tr>
<tr><td rowspan="2">Исходные данные</td><td>Компоновщик (редактор связей)</td><td rowspan="2">Построение исполнительного модуля</td></tr>
<tr><td>Загрузочный модуль</td></tr>
<tr><td rowspan="2">Выполнение</td><td>Загрузчик</td><td rowspan="2">Загрузка, настройка в памяти ПК</td></tr>
<tr><td>Абсолютный модуль</td></tr>
<tr><td colspan="2"></td><td>Результат  </td></tr>
</table>

Рисунок 16. Разработка программы в системе программирования

2. Жизненный цикл программного обеспечения

Жизненный цикл программного обеспечения - это процесс его создания и применения от начала до конца. Этот процесс состоит из нескольких стадий: определения требований и спецификаций, проектирования, программирования, отладки и сопровождения.

Первая стадия – определение требований и спецификаций, может быть названа стадией системного анализа. На ней устанавливаются общие требования к ПО: по надежности, технологичности, правильности, универсальности, эффективности, информационной согласованности и т.д. Они дополняются требованиями заказчика, включающими пространственно-временные ограничения, необходимые функции и возможности, режимы функционирования, требования точности и надежности и.т.д., т.е. вырабатывается описание системы с точки зрения пользователя. Итогом выполнения этого этапа являются эксплуатационные и функциональные спецификации, содержащие конкретное описание ПО. Эксплуатационные спецификации содержат сведения о быстродействии ПО, затратах памяти, требуемых технических средствах, надежности и т.д. Функциональные спецификации определяют функции, которые должно выполнять ПО, т.е. в них определяется, что надо делать системе, а не то, как это делать.

Второй стадией является проектирование ПО. На этом этапе:

1. Формируется структура ПО и разрабатываются алгоритмы, задаваемые спецификациями.
2. Устанавливается состав модулей с разделением их на иерархические уровни на основе изучения схем алгоритмов.
3. Выбирается структура информационных массивов.
4. Фиксируются межмодульные интерфейсы.

Цель этапа – иерархическое разбиение сложных задач создания ПО на подзадачи меньшей сложности. Результатом работы на этом этапе являются спецификации на отдельные модули, дальнейшая декомпозиция которых нецелесообразна.

Третья стадия – программирование. На данном этапе производится программирование модулей. Этап менее сложен по сравнению со всеми остальными. Проектные решения, полученные на предыдущей стадии, реализуются в виде программ. Разрабатываются отдельные блоки и подключаются к создаваемой системе. Одна из задач – обоснованный выбор языков программирования. На этой же стадии решаются все вопросы, связанные с особенностями типа ЭВМ,

Четвертая стадия – отладка ПО заключается в проверке выполнения всех требований, всех структурных элементов системы на таком количестве всевозможных комбинаций данных, какое только позволяют здравый смысл и бюджет. Этап предполагает выявление в программах ошибок, проверку работоспособности ПО, а также его соответствие спецификациям.

Пятая стадия – сопровождение, т.е. процесс исправления ошибок, координации всех элементов системы в соответствии с требованиями пользователя, внесения всех необходимых ему исправлений и изменений. Он вызван, как минимум, двумя причинами: во-первых, в ПО остаются ошибки, не выявленные при отладке; во-вторых, пользователи в ходе эксплуатации ПО настаивают на внесении в него изменений и его дальнейшем совершенствовании.

В общем случае, в процессе разработки ПО требуется осуществить несколько итераций (последовательных приближений) к приемлемому варианту системы.

Таблица 2 Стадии жизненного цикла ПО

Стадии жизненного цикла программного обеспечения	**Количество ошибок**	**Обнаружение ошибок, %**	**Затраты на устранение ошибок, %**
Определение требований и спецификаций			9
Проектирование	61-64		6
Программирование	36-69		10
Отладка		46	12

Сопровождение		54	63

Исходя из этого, возникает ряд требований к инженерному программированию. Они состоят в необходимости разработки и сопровождения ПО, которой гарантирует, что все стадии являются:

- Исключительно надежными;
- Удобными в использовании;
- Естественными;
- Проверяемыми;
- Труднодоступными для злоупотреблений;
- Оставляющими главную роль за человеком, а не за машиной.

Попытка сформулировать общую цель инженерного программирования, достижение которой привело бы к удовлетворению всех указанных требований, обречена на неудачу, так как некоторые требования противоречат друг другу.

Контрольные вопросы

1. Перечислить и описать этапы разработки программы в системе программирования.
2. Что такое объектный модуль?
3. Что такое абсолютный модуль?
4. Какие функции должен выполнить этап трансляции программы?
5. Что такое препроцессинг?
6. Перечислить стадии жизненного цикла ПО.
7. На какой стадии жизненного цикла программного обеспечения обнаруживают самое большое количество ошибок?

Тема 6. Общие принципы разработки ПО. Жизненный цикл программного обеспечения

План занятия

1. Общие принципы разработки программного обеспечения

2. Общесистемные принципы разработки программного обеспечения

1. Общие принципы разработки программного обеспечения

Программное обеспечение различается по назначению, выполняемым функциям, формам реализации. В этом смысле ПО – сложная, достаточно уникальная система. Однако существуют некоторые общие принципы, которые следует использовать при разработке ПО.

Частотный принцип. Основан на выделении в алгоритмах и в обрабатываемых структурах действий и данных по частоте использования. Для действий, которые часто встречаются при работе ПО, обеспечиваются условия их быстрого выполнения. К данным, к которым происходит частое обращение, обеспечивается наиболее быстрый доступ, а подобные операции стараются сделать более короткими.

Принцип модульности. Под модулем в общем случае понимают функциональный элемент рассматриваемой системы, имеющий оформление, законченное и выполненное в пределах требований системы, и средства сопряжения с подобными элементами или элементами более высокого уровня данной или другой системы. Способы обособления составных частей ПО в отдельные модули могут быть различными. Чаще всего разделение происходит по функциональному признаку. В значительной степени разделение системы на модули определяется используемым методом проектирования ПО.

Принцип функциональной избирательности. Этот принцип является логическим продолжением частотного и модульного принципов и используется при проектировании ПО, объем которого существенно превосходит имеющийся объем оперативной памяти. В ПО выделяется некоторая часть важных модулей, которые постоянно должны быть в состоянии готовности для эффективной организации вычислительного процесса. Эту часть в ПО называют ядром или монитором. При формировании состава монитора требуется удовлетворить два противоречивых требования. В состав монитора, помимо чисто управляющих модулей, должны войти наиболее часто используемые модули. Количество модулей должно быть таким, чтобы объем памяти, занимаемой монитором, был не слишком большим. Программы, входящие в состав монитора, постоянно находятся в оперативной памяти. Остальные части ПО постоянно хранятся во внешних запоминающих устройствах и загружаются в оперативную память только при необходимости, иногда перекрывая друг друга.

Принцип генерируемости. Данный принцип определяет такой способ исходного представления ПО, который бы позволял осуществлять настройку на конкретную конфигурацию технических средств, круг решаемых проблем, условия работы пользователя.

Принцип функциональной избыточности. Этот принцип учитывает возможность проведения одной и той же работы (функции) различными средствами. Особенно важен учет этого принципа при разработке

пользовательского интерфейса для выдачи данных из-за психологических различий в восприятии информации.

Принцип «по умолчанию». Применяется для облегчения организации связей с системой как на стадии генерации, так и при работе с уже готовым ПО. Принцип основан на хранении в системе некоторых базовых описаний структур, модулей, конфигураций оборудования и данных, определяющих условия работы с ПО. Эту информацию ПО использует в качестве заданной, если пользователь забудет или сознательно не конкретизирует ее. В данном случае ПО само установит соответствующие значения.

2. Общесистемные принципы

При создании и развитии ПО рекомендуется применять следующие общесистемные принципы:

- Принцип включения, который предусматривает, что требования по созданию, функционированию и развитию ПО определяются со стороны более сложной, включающей его в себя системы;
- Принцип системного единства, который состоит в том, что на всех стадиях созданий, функционирования и развития ПО его целостность будет обеспечиваться связями между подсистемами, а также функционированием подсистемы управления;
- Принцип развития, который предусматривает в ПО возможность его наращивания и совершенствования компонентов и связей между ними;
- Принцип комплексности, который заключается в том, что ПО обеспечивает связность обработки информации как отдельных элементов, так и для всего объема данных в целом на всех стадиях обработки;
- Принцип информационного единства, определяющий, что во всех подсистемах, средствах обеспечения и компонентах ПО используются единые термины, символы, условные обозначения и способы представления;
- Принцип совместимости, который состоит том, что язык, символы, коды и средства обеспечения ПО согласованы, обеспечивают совместное функционирование всех его подсистем и сохраняют открытой структуру системы в целом;
- Принцип инвариантности, который предопределяет, что подсистемы и компоненты программного обеспечения инвариантны к обрабатываемой информации, т.е. являются универсальными или типовыми.

Контрольные вопросы

1. Перечислить и описать общие принципы разработки ПО.
2. Какой принцип учитывает возможность проведения одной и той же работы (функции) различными средствами.
3. Какой принцип позволяет выделить в алгоритмах и в обрабатываемых структурах действия и данные по частоте использования.
4. Какие существуют общесистемные принципы разработки ПО?

Тема 7. Структурное программирование

План занятия

1. Понятие технологии структурного программирования
2. Теорема о структурном программировании
3. Преимущества структурного программирования

1. Понятие технологии структурного программирования

В истории применения компьютеров вычислительная техника все время используется на пределе своих возможностей. Каждое новое достижение в аппаратном либо в программном приводит к попыткам расширить сферу применения ЭВМ, что влечет за собой постановку новых задач, для решения которых, в свою очередь, нужны новые вычислительные средства.

Основа для массового промышленного программирования была создана с разработкой методов построения программ.

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э. Дейкстрой, разработана и дополнена Н. Виртом.

В соответствии с данной методологией

1. Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:
   - **последовательное исполнение** — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;
   - **ветвление** — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;
   - **цикл** — многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

   В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.
2. Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки)

могут оформляться в виде т. н. подпрограмм(процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция **вызова подпрограммы**. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.

3. Разработка программы ведётся пошагово, методом «сверху вниз».

Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются «заглушки», которые ничего не делают. Полученная программа проверяется и отлаживается. После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы. Разработка заканчивается тогда, когда не останется ни одной «затычки», которая не была бы удалена. Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов, и может быть уверен, что общая структура всех более высоких уровней программы верна. При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения, и они вносятся, не затрагивая части программы, непосредственно не связанные с ними. Это позволяет гарантировать, что при внесении изменений и исправлении ошибок не выйдет из строя какая-то часть программы, находящаяся в данный момент вне зоны внимания программиста.

2. **Теорема о структурном программировании**

Теорема Бома-Якопини

Любую схему алгоритма можно представить в виде композиции вложенных блоков begin и end, условных операторов if, then, else, циклов с предусловием (while) и может быть дополнительных логических переменных (флагов).

Эта теорема была сформулирована итальянскими математиками К. Бомом и Дж. Якопини в 1966 году и говорит нам о том, как можно избежать использования оператора перехода goto.

Методология структурного программирования появилась как следствие возрастания сложности решаемых на компьютерах задач, и соответственного усложнения программного обеспечения. В 70-е годы XX века объёмы и сложность программ достигли такого уровня, что «интуитивная» (неструктурированная, или «рефлекторная») разработка программ, которая была нормой в более раннее время, перестала удовлетворять потребностям практики. Программы становились слишком сложными, чтобы их можно было

нормально сопровождать, поэтому потребовалась какая-то систематизация процесса разработки и структуры программ.

Наиболее сильной критике со стороны разработчиков структурного подхода к программированию подвергся оператор GOTO (оператор безусловного перехода), имевшийся тогда почти во всех языках программирования. Неправильное и необдуманное использование произвольных переходов в тексте программы приводит к получению запутанных, плохо структурированных программ (т. н. спагетти-кода), по тексту которых практически невозможно понять порядок исполнения и взаимозависимость фрагментов.

Следование принципам структурного программирования сделало тексты программ, даже довольно крупных, нормально читаемыми. Серьёзно облегчилось понимание программ, появилась возможность разработки программ в нормальном промышленном режиме, когда программу может без особых затруднений понять не только её автор, но и другие программисты. Это позволило разрабатывать достаточно крупные для того времени программные комплексы силами коллективов разработчиков, и сопровождать эти комплексы в течение многих лет, даже в условиях неизбежных изменений в составе персонала.

Методология структурной разработки программного обеспечения была признана «самой сильной формализацией 70-х годов». После этого слово «структурный» стало модным в отрасли, и его начали использовать везде, где надо и где не надо. Появились работы по «структурному проектированию», «структурному тестированию», «структурному дизайну» и так далее. В общем, произошло примерно то же самое, что происходило в 90-х годах и происходит в настоящее время с терминами «объектный», «объектно-ориентированный» и «электронный».

Преимущества структурного программирования

Перечислим некоторые достоинства структурного программирования:

1. Структурное программирование позволяет значительно сократить число вариантов построения программы по одной и той же спецификации, что значительно снижает сложность программы и, что ещё важнее, облегчает понимание её другими разработчиками.
2. В структурированных программах логически связанные операторы находятся визуально ближе, а слабо связанные — дальше, что позволяет обходиться без блок-схем и других графических форм изображения алгоритмов (по сути, сама программа является собственной блок-схемой).
3. Сильно упрощается процесс тестирования и отладки структурированных программ.

Цель структурного программирования — повысить производительность труда программистов, в том числе при разработке больших и сложных программных комплексов, сократить число ошибок,

упростить отладку, модификацию и сопровождение программного обеспечения.

Следование принципам структурного программирования сделало тексты программ, даже довольно крупных, нормально читаемыми. Серьёзно облегчилось понимание программ, появилась возможность разработки программ в нормальном промышленном режиме, когда программу может без особых затруднений понять не только её автор, но и другие программисты. Это позволило разрабатывать достаточно крупные для того времени программные комплексы силами коллективов разработчиков, и сопровождать эти комплексы в течение многих лет, даже в условиях неизбежной ротации кадров.

Структурный подход к программированию как раз и имеет целью снижение трудоемкости всего процесса создания программного обеспечения от технического задания на разработку до завершения эксплуатации. Он означает необходимость единой дисциплины на всех стадиях разработки программы. В понятие структурного подхода к программированию обычно включают нисходящие методы разработки программ (принцип «сверху вниз»), собственно структурное программирование и так называемый сквозной структурный контроль.

Основной целью структурного программирования является уменьшение трудностей тестирования и доказательства правильности программы. Это особенно важно при разработке больших программных систем. Опыт применения методов структурного программирования при разработке ряда сложных операционных систем показывает, что правильность логической структуры системы поддается доказательству, а сама программа допускает достаточно полное тестирование. В результате в готовой программе встречаются только тривиальные ошибки кодирования, которые легко исправляются. Структурное программирование улучшает ясность и читабельность программ. Программы, которые написаны с использованием традиционных методов, особенно те, которые перегружены операторами GOTO, имеют хаотичную структуру. Структурированные программы имеют последовательную организацию, поэтому возможно читать такую программу сверху донизу без перерыва. Наконец, структурное программирование призвано улучшить эффективность программ.

Контрольные вопросы

1. Какая технология программирования называется структурной.

2. Преимущества использования технологии структурного программирования.
3. Недостатки технологии структурного программирования.
4. Теорема Бома-Якопини.

Тема 8. Технология объектно-ориентированного программирования

План занятия

1. Объектно-ориентированная технология программирования: определение, свойства.
2. Преимущества объектно- ориентированного подхода
3. Недостатки технологии объектно-ориентированного программирования

1. Объектно-ориентированная технология программирования: определение, свойства

Объектно-ориентированное программирование (ООП) представляет собой способ программирования, который напоминает процесс человеческого мышления. ООП более структурированный, чем другие способы программирования и позволяет создавать модульные программы с представлением данных на определенном уровне абстракции. Объектно-ориентированный язык программирования характеризуется тремя основными свойствами:

- Инкапсуляция;
- Наследование;
- Полиморфизм.

Рассмотрим пример построения изображения «звездного неба» в двумерной проекции. Пусть звезда в нашем определении – это точка на экране с координатами X, Y, которая имеет цвет, может быть видимой или невидимой и может перемещаться по небу. Планета – это цветной круг с координатами центра в точке X, Y, который тоже может быть видимым или невидимым и должен уметь перемещаться. Все звезды и планеты в нашем примере однотипны и могут представляться одинаковыми свойствами и процедурами обработки, т.е. для решения задачи необходимо создать так называемые объекты типа «Звезда» и «Планета». Таким образом, объект – это понятие, сочетающее в себе совокупность данных и действий над ними. Свойства – это характеристики состояния объекта, а действия над данными объекта называются методами.

Основой изображения любого элемента является его положение (позиция) на экране (например, значения координат X и Y относительно

левого верхнего угла экрана). Опишем объект «Позиция» с координатами X и Y и методом «Назначить координаты»:

Позиция (X, Y, Назначить XY)

Рассмотрим теперь объект «Звезда», который по нашему определению может быть описан следующим образом:

Звезда (X, Y, Видимость, Цвет, Назначить XY, Назначить цвет, Зажечь, Погасить, Переместить)

Обратим внимание на то, что возможности объекта Позиция полностью входят в объект Звезда. Такое свойство объектов называется наследованием и позволяет описать объект Звезда с учетом наследования возможностей объекта Позиция:

Звезда (Позиция, Видимость, Цвет, Назначить цвет, Зажечь, Погасить, Переместить)

Наследование позволяет повторно использовать уже созданную часть программного кода в других проектах. Таким образом, наследование – это определение объекта и дальнейшее использование всех его свойств для построения иерархии порожденных объектов с возможностью для каждого порожденного объекта, относящегося к иерархии, доступа к коду и данным всех порождающих объектов. Т.е. объекты наследуют характеристики и поведение других объектов, называемых порождающими или родительскими.

Инкапсуляция. Объекты являются высшим уровнем абстракции данных. Объект можно разделить на части, но тогда он перестанет быть объектом. Отношения частей к целому и взаимоотношения между частями становятся понятнее тогда, когда все содержится вместе как единое целое. Это называется инкапсуляцией. Таким образом, объединение в одном месте всех данных и методов объекта (включая данные и методы объектов-предков) называется инкапсуляцией.

Полиморфизм – присваивание определенному действию одного имени, которое затем совместно используется по всей иерархии объектов сверху донизу, причем каждый объект иерархии выполняет это действие характерным именно для него способом.

Например, объект Планета, должен наследовать все возможности объекта Звезда, но методы Назначить цвет, Зажечь, Погасить, Переместить, по результату действия являясь теми же самыми, фактически не могут быть реализованы одинаковой последовательностью команд в случае со Звездой и Планетой. Эта ситуация разрешается следующим образом: методы объектов имеют одинаковое имя, а внутреннее описание методов будет различно для разных объектов. Такое свойство называется полиморфизмом, а объект Планета имеет следующее описание:

Планета (Звезда, радиус, назначить цвет, Зажечь, Погасить, Переместить)

Полиморфизм основывается на возможности включения в данные объекта также и информации о методах обработки этих данных. При этом различные объекты используют одинаковую абстракцию, т.е. могут обладать свойствами и методами с одинаковыми именами.

Переход от структурно-процедурного подхода к объектно-ориентированному программированию, подобно переходу от низкоуровневых языков программирования к языкам высокого уровня, требует значительных затрат на обучение. Естественно, что платой за это является повышение производительности труда программистов при проектировании и реализации программного обеспечения. Другое преимущество ООП перед императивным подходом – более высокий процент повторного использования уже разработанного программного кода.

При этом, в отличие от предыдущих подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или, иначе, концепций, на которых он базируется..

2. Преимущества объектно- ориентированного подхода

1. *Объектная декомпозиция* дает возможность создавать программные системы меньшего размера путем использования общих механизмов, обеспечивающих необходимую экономию выразительных средств. Использование объектного подхода существенно повышает уровень унификации разработки и пригодность для повторного использования не только программ, но и проектов, что в конце концов ведет к созданию среды разработки и переходу к сборочному созданию ПО. Системы зачастую получаются более компактными, чем их структурные эквиваленты, что означает не только уменьшение объема программного кода, но и удешевление проекта за счет использования предыдущих разработок.

2. *Объектная декомпозиция* уменьшает риск создания сложных систем ПО, так как она предполагает эволюционный путь развития системы на базе относительно небольших подсистем. Процесс интеграции системы растягивается на все время разработки, а не превращается в единовременное событие.

3. *Объектная модель вполне естественна*, поскольку в первую очередь ориентирована на человеческое восприятие мира, а не на компьютерную реализацию.

4. *Объектная модель* позволяет в полной мере использовать выразительные возможности объектных и объектно-ориентированных языков программирования.

3. Недостатки технологии объектно-ориентированного программирования

К недостаткам объектно-ориентированного подхода относятся некоторое снижение производительности функционирования ПО и высокие начальные затраты. Объектная декомпозиция существенно отличается от функциональной, поэтому переход на новую технологию связан как с преодолением психологических трудностей, так и дополнительными

финансовыми затратами. Безусловно, объектно-ориентированная модель наиболее адекватно отражает реальный мир, представляющий собой совокупность взаимодействующих (посредством обмена сообщениями) объектов.

При переходе от структурного подхода к объектному, как при всякой смене технологии, необходимо вкладывать деньги в приобретение новых инструментальных средств. Здесь следует учесть и расходы на обучение (овладение методом, инструментальными средствами и языком программирования). Для некоторых организаций эти обстоятельства могут стать серьезными препятствиями.

Объектно-ориентированный подход не дает немедленной отдачи. Эффект от его применения начинает сказываться после разработки двух-трех проектов и накопления повторно используемых компонентов, отражающих типовые проектные решения в данной области. Переход организации на объектно-ориентированную технологию – это смена мировоззрения, а не просто изучение новых CASE-средств и языков программирования.

Контрольные вопросы

5. Какая технология программирования называется объектно-ориентированной.
6. Преимущества использования технологии объектно-ориентированного программирования.
7. Недостатки технологии ООП.
8. Свойства объектно-ориентированного языка программирования.

Задания для самостоятельного решения

1. Составить словесно-формульное описание алгоритма: имеются два кувшина емкостью 3 и 8 литров. Требуется набрать 7л воды.

2. Составить блок-схему алгоритма вычисления площади треугольника по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

,

где a,b,c – стороны треугольника

p- полупериметр

3. Составить блок-схему алгоритма: определить площадь трапеции по введенным значениям оснований (a и b) и высоты (h).

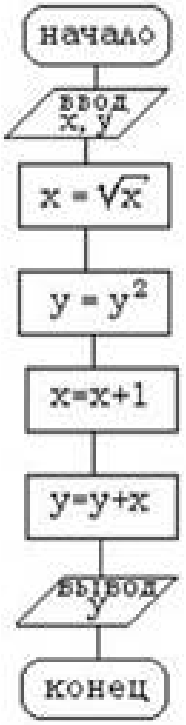
4. Составить блок-схему алгоритма для решения квадратного уравнения: $ax^2 + bx + c = 0$.

5. Даны числа a, b . Известно, что число a меняется от -10 до 10 с шагом 5, $b = 7$ и не изменяется. Вычислить сумму S и разность R чисел a и b для всех значений a и b . Составить блок-схему алгоритма.

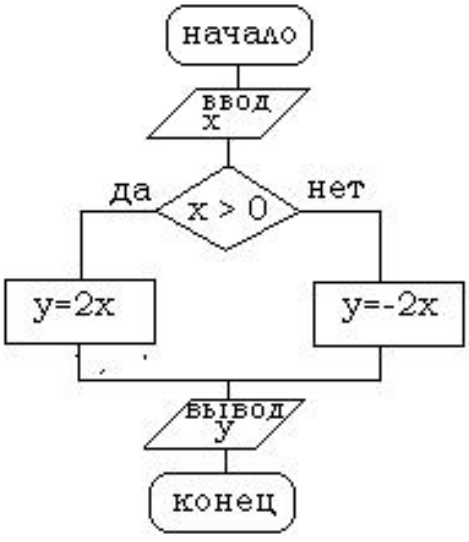
6. Леонардо из Пизы, известный как Фибоначчи, был первым из великих математиков Европы позднего Средневековья. Числовой ряд, который называется его именем, получился в результате решения задачи о кроликах, которую Фибоначчи изложил в своей «Книге Абака», написанной в 1202 году. Он выглядит так:
   1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...
   В этом ряду каждое следующее число, начиная с третьего, равно сумме двух предыдущих. Составить словесный алгоритм и блок-схему проверки принадлежности введенного числа n ряду Фибоначчи.

7. Составить блок-схему алгоритма нахождения наибольшего числа из 3-х введенных чисел.

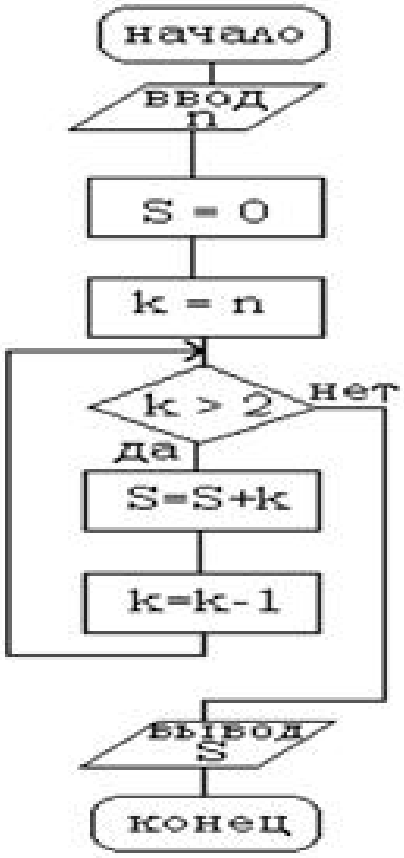
8. Определить результат выполнения алгоритма при определённых значениях исходных данных. Например, при x=16 и y=2



9. Дана блок-схема алгоритма. Определить результат выполнения алгоритма при определённых значениях исходных данных.
Например, при x= - 6 или x=0 или x=7



10. Дана блок-схема алгоритма. Определить результат выполнения алгоритма при определённых значениях исходных данных. Например, при n=4 или n=1



11. Подготовить реферат на тему «История создания языка программирования (выбор языка программирования по своему усмотрению)».

12. Подготовить реферат на тему «Интегрированные среды программирования».

Список рекомендуемой литературы

Основные источники:

1. Кривцов, А.Н. Алгоритмизация и программирование. Основы программирования на С/С++: учебное пособие / А.Н.Кривцов . С.В. Хорошенко . – СПб.: Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, 2020. – 202 с. – (Бакалавриат)
2. Бедердинова, О.И. Основы алгоритмизации и структурного программирования: Учебное пособие / О.И. Бедердинова . – СПб.:Северный (Арктический) федеральный университет имени М. В. Ломоносова, 2019. – 88 с. – (Бакалавриат)
3. Курбанисмаилов, З. М Основы языка программирования С#: Учебно-методическое пособие /З.М. Курбанисмаилов , Е.В. Кашкин . – М.:МИРЭА , 2019. – 93 с. – (Бакалавриат, Специалитет)
4. Тюкачев, Н. А. С#. Основы программирования: Учебное пособие для СПО / Н.А. Тюкачев, В.Г. Хлебостроев. – М.: Лань, 2022. – 272 с.
5. Залогова, Л.А. Основы объектно-ориентированного программирования на базе языка С# / Л.А. Залогова. – 2-е изд., стер. – М.: Лань , 2021. – 192 с. – (Среднее профессиональное образование)

Дополнительные источники:

1. Агальцов В.П. Математические методы в программировании: учебник. – 2-е изд., перераб. И доп. –М.: ИД «ФОРУМ», 2013. -240 с.
2. Джеймс М. ЛэйсиVisualC++ 6 Distributed ,Санкт-Петербург, «Питер», 2014г. - 678с.
3. Казиев В.М. Введение в информатику. Раздел (лекция) 1 - Введение. История, предмет, структура информатики. Интернет-Университет информационных технологий, 2014. – 264 с..
4. Климова Л.М. "Практическое программирование. Решение типовых задач. С/С++". – М: Кудиц-образ, 2013. – 596 с.
5. Мейер Б., Бодуэн К.. Методы программирования: В 2-х томах. М.: “Мир”, 2014г.- 642 с.

Электронные источники:

1. Деревягос С. С++ 3rd: комментарии http://lib.ru/CTOTOR/cpp3comm.txt
2. Страуструп Б. Введение в язык С++http://lib.ru/CPPHB/cpptut.txt
3. Страуструп Б. Справочное руководство по С++http://lib.ru/CPPHB/cppref.txt