НАРОДНАЯ ОБРАЗОВАНИЯ РЕСПУБЛИКА
УЗБЕКМСТАН
Джизакский Государственный Педагогический Институт
Им.А.Кадырий

С.М.ЖУМАБОЕВ, Г.И.НАБИРАЕВА

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

ДЖИЗАК 2005

С.М.ЖУМАБОЕВ, Г.И.НАБИРАЕВА

Основы алгоритмизации и программирования 42ст.

Информатика как общеобразовательная дисциплина прочно вошла в образовательные стандарты большинства специальностей высших и средних учебных заведений. По количеству студентов и учащихся, изучающих информатику, эта дисциплина является одной из самых популярных.

Несмотря на определенные отличия содержания учебных программ по информатике, в этом пособие можно выделить некоторый инвариант, изучаемый всеми категориями учащихся. Предлагаемый учебная пособия по информатике является попыткой реализации указанного инварианта. Его можно использовать как основу курса для большинства специальностей высших и средних учебных заведений. Учебная пособия является не просто сборником теоретических сведений, но и платформой для формирования навыков и умений (лабораторные работы), проверки знаний (тестовая система, контрольные вопросы).

Среди особенностей учебная пособия можно выделить следующие:

- для каждого тематического раздела составлены контрольные вопросы;
- вопросы могут быть изменены преподавателем;
- учебная пособия ориентирован на использование стандартного программного обеспечения, которое имеется в большинстве учебных заведений.

Отв.редактор к.ф.н.У.Каситов.

Рецензенты:
Д.т.н. Х.О.Туракулов, к.т.н. Р.М.Юсупов.

Утверждено к печати
Ученым советом Института ГДЖПИ.

Формат бумаги А4 (210х297), 1,2

Печет плоская. Тираж 300 экз. Заказ 25.

Цена договорная.

Издательство "Педагог" ДЖГПИ.

1.Основы алгоритмизации и программирования

Программа — это детальное и законченное описание алгоритма средствами языка программирования. Исполнителем программы является компьютер. Для выполнения компьютером программа должна быть представлена в машинном коде — последовательности чисел, понимаемых процессором. Написать программу в машинных кодах вручную достаточно сложно. Поэтому сегодня практически все программы создаются с помощью языков программирования, которые по своим синтаксису и семантике приближены к естественному человеческому языку. Это снижает трудоемкость программирования. Однако, текст программы, записанный с помощью языка программирования, должен быть преобразован в машинный код. Эта операция выполняется автоматически с помощью специальной служебной программы, называемой **транслятором**.

Трансляторы делятся на два типа: **интерпретаторы** и **компиляторы**.

Интерпретатор переводит в машинный код и выполняет очередной оператор (команду) программы. Если команда повторяется, то интерпретатор рассматривает ее как встреченную впервые.

Компилятор переводит в машинный код исходный текст программы целиком. Поэтому достоинство компиляторов — быстродействие и автономность получаемых программ. Достоинство интерпретаторов — их компактность, возможность остановить в любой момент выполнение программы, выполнить различные преобразования данных и продолжить работу программы.

Примерами служебных программ — интерпретаторов являются GW Basic, Лого, школьный алгоритмический язык, многие языки программирования баз данных. Компиляторами являются Turbo Pascal, C++, Delphi.

Средства создания программ

В общем случае для создания программ нужно иметь следующие компоненты

- текстовый редактор — для набора исходного текста программы;
- компилятор — для перевода текста программы в машинный код;
- редактор связей — для сборки нескольких откомпилированных модулей в одну программу;

- библиотеки функций — для подключения стандартных функций к программе.

Современные системы программирования включают в себя все указанные компоненты и называются **интегрированными системами**.

Исходный текст программы можно получить без записи его вручную в текстовом редакторе. Существуют системы визуального программирования — **RAD**-среды (Rapid Application Development), которые, не исключая возможности записи программы вручную, позволяют создавать текст программы автоматически, путем манипуляций со стандартными элементами управления, включенными в RAD-среду. Поэтому для RAD-среды понятие «программирование» часто заменяют понятием «проектирование».

По способу разработки программ можно выделить два подхода:

- **процедурное программирование** — это программирование, при котором выполнение команд программы определяется их последовательностью, командами перехода, цикла или обращениями к процедурам;
- **объектно-ориентированное программирование** – программирование, при котором формируются программные объекты, имеющие набор свойств, обладающие набором методов и способные реагировать на события, возникающие как во внешней среде, так и в самом объекте (нажатие мыши, срабатывание таймера, превышение числовой границы и т.д.). Таким образом, выполнение той или иной части программы зависит от событий в программной системе.

Объектно-ориентированное программирование (ООП) не исключает, а охватывает технологию процедурного программирования.

1.1 Основные системы программирования

Из универсальных языков программирования наиболее популярны следующие: Basic; Pascal; C++; Java.

Для языка Basic существует много версий, реализованных и как интерпретаторы и как компиляторы. В России Basic традиционно используется в курсе информатики средней школы. Среда визуального программирования Microsoft Visual Basic используется как программная поддержка приложений MS Office.

Язык Pascal является компилируемым и широко используется как среда для обучения программированию в ВУЗах. RAD-средой, наследующей его основные свойства, является среда Borland Delphi.

Для языка C++ RAD-средой является Borland C++ Builder. Этот компилируемый язык часто используется для разработки программных приложений, в которых необходимо обеспечить быстродействие и экономичность программы.

Язык Java — интерпретируемый язык — позволяет создавать платформно-независимые программные модули, способные работать в компьютерных сетях с различными операционными системами. RAD-средой для него является Symantec Cafe.

1.2 Основные этапы развития языков программирования

Языки программирования развивались одновременно с развитием ЭВМ. С начала 50-х годов это были низкоуровневые языки (машинные и ассемблеры). В 1956 году появился язык Фортран, а в 1960 — Алгол-60. Это языки компилирующего типа, существенно уменьшившие трудоемкость программирования. Языки ориентированы на выполнение математических вычислений. В дальнейшем возникло большое количество различных языков, претендовавших на универсальность (PL/1) или для решения конкретных задач (COBOL — для деловых задач, ЛОГО — для обучения, Пролог — для разработки систем искусственного интеллекта). С середины 60-х до начала 80-х разработаны и получили распространение языки Pascal, Basic, Си, Ада и другие.

Принципиально новым этапом в развитии языков программирования стало появление методологии непроцедурного (ООП) программирования (см. выше). Основные достоинства ООП — быстрота разработки интерфейса программного приложения, возможность наследования свойств программных объектов.

Основы алгоритмизации

Алгоритм — это предписание некоторому **исполнителю** выполнить конечную последовательность действий, приводящую к некоторому результату.

В программировании алгоритм является фундаментом программы, а основным **исполнителем** — компьютер. На стадии тестирования алгоритма исполнителем может быть сам программист.

Алгоритм может быть записан с помощью **блок-схемы**, текстовым предписанием, с помощью рисунков, таблично или на специальном алгоритмическом языке. Наиболее популярны блок-схемы и предписания. Преимущество блок-схем — в наглядности алгоритма.

Основными свойствами алгоритма являются:

- **дискретность** — представление алгоритма в виде последовательности шагов;
- **массовость** — применимость алгоритма к некоторому множеству исходных данных;
- **определенность** — за конечное число шагов либо должен быть получен результат, либо доказано его отсутствие;
- **однозначность** — при повторном применении алгоритма к тем же исходным данным должен быть получен тот же результат.

Из перечисленных свойств лишь дискретность является обязательным свойством алгоритма. Можно привести примеры, когда невыполнение свойств массовости, определенности и однозначности не позволяет говорить об отсутствии алгоритма.

Рис. 1. Типовые блоки

Для изображения алгоритмов будем использовать блок-схемы, формируемые из типовых блоков, показанных на рис. 1.

В теории алгоритмов доказано, что любой, сколь угодно сложный алгоритм может быть составлен из трех основных алгоритмических структур: линейной, ветвления и цикла, показанных, соответственно на рис. 2, 3, 4.

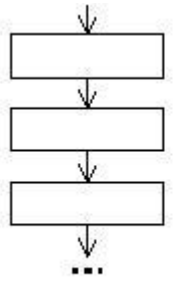


Рис.2. Линейная структура

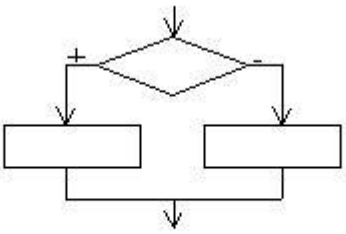


Рис.3. Ветвление

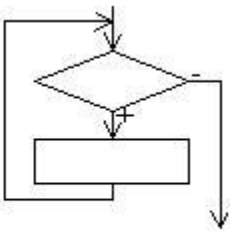


Рис.4. Цикл

Линейная структура предполагает последовательное выполнение действий, без их повторения или пропуска некоторых действий. Обычно программисты стремятся к тому, чтобы алгоритм имел линейную структуру.

Структура "ветвление" предполагает выполнение одной из двух групп действий в зависимости от выполнения условия в блоке ветвления. На рис. 3 знаком "+" показано выполнение условия, а знаком "-" — его невыполнение. Часто используется неполная команда ветвления, когда один из блоков действия отсутствует.

Структура "цикл" имеет несколько разновидностей. На рис. 4 показан цикл типа "пока" с предусловием. Действия внутри этого цикла повторяются пока выполняется условие в блоке ветвления, причем сначала проверяется условие, а затем выполняется действие. Достаточно часто используются другие типы цикла, показанные на рис. 5 и 6.

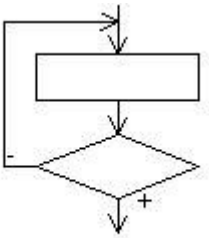


Рис.5. Цикл "пока" с постусловием

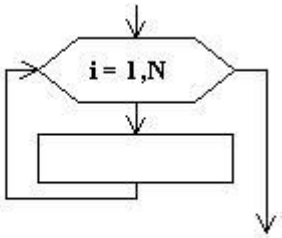


Рис.6. Цикл "для"

В цикле с постусловием проверка условия выхода из цикла выполняется после очередного действия. Цикл "для" является модификацией цикла "пока" для ситуации, когда заранее известно количество повторений некоторых действий. Запись в блоке заголовка цикла на рис.6 показывает пример описания заголовка цикла, в котором действия повторяются столько раз, сколько целых значений приобретает параметр цикла i от своего начального значения 1 до конечного N с шагом 1. Обычно шаг не указывается, если он равен 1.

В языках программирования имеются команды, реализующие показанные выше структуры.

При разработке блок-схемы допускается делать любые записи внутри блоков, однако эти записи должны содержать достаточно информации для выполнения очередных действий.

Пример 1

Разработать блок-схему алгоритма Евклида, определяющего наибольший общий делитель (НОД) двух натуральных чисел A и B.

В основе алгоритма Евклида лежит правило:

НОД(A,B)= НОД(min(A,B), |A-B|),

где НОД(A,B) — наибольший общий делитель двух натуральных чисел A и B.

Основной идеей решения задачи является многократное применение указанного выше правила, после которого большее из чисел очередной пары уменьшается. Решение получено, когда числа оказываются равны друг другу. Поскольку количество повторений заранее неизвестно, в алгоритме следует применить цикл "пока" с предусловием (рис. 7).

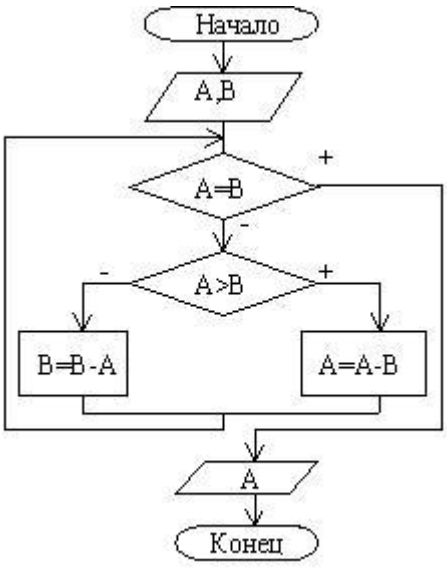


Рис.7. Алгоритм Евклида

1.3 Методика разработки алгоритмов

Разработке алгоритма предшествуют такие этапы, как формализация и моделирование задачи. **Формализация** предполагает замену словесной формулировки решаемой задачи краткими символьными обозначениями, близкими к обозначениям в языках

программирования или к математическим. **Моделирование** задачи является важнейшим этапом, целью которого является поиск общей концепции решения. Обычно моделирование выполняется путем выдвижения гипотез решения задачи и их проверке любым рациональным способом (прикидочные расчеты, физическое моделирование и т.д.). Результатом каждой проверки является либо принятие гипотезы, либо отказ от нее и разработка новой.

При разработке алгоритма используют следующие основные принципы.

1. **Принцип поэтапной детализации алгоритма** (другое название — "проектирование сверху-вниз"). Этот принцип предполагает первоначальную разработку алгоритма в виде укрупненных блоков (разбиение задачи на подзадачи) и их постепенную детализацию.
2. **Принцип "от главного к второстепенному"**, предполагающий составление алгоритма, начиная с главной конструкции. При этом, часто, приходится "достраивать" алгоритм в обратную сторону, например, от середины к началу.
3. **Принцип структурирования**, т.е. использования только типовых алгоритмических структур при построении алгоритма. Нетиповой структурой считается, например, циклическая конструкция, содержащая в теле цикла дополнительные выходы из цикла. В программировании нетиповые структуры появляются в результате злоупотребления командой безусловного перехода (GoTo). При этом программа хуже читается и труднее отлаживается.

Говоря о блок-схемах, как о средстве записи алгоритма, можно дать еще один совет по их разработке. Рекомендуется после внесения исправлений в блок-схему аккуратно перерисовывать ее с учетом этих исправлений. Аккуратность записи есть аккуратность мысли программиста. Аккуратно записанный и детализованный алгоритм упрощает его программирование и отладку.

1.4 Основные этапы компьютерного решения задач

Обобщим рассмотренные выше примеры и принципы разработки алгоритмов и программ и выделим главные этапы методики программирования задач.

1. **Постановка задачи**. Основное требование к постановке задачи – достаточное количество информации для решения задачи. Очень часто постановка задачи выполняется не программистом, а некоторым Заказчиком. Программист является

Исполнителем заказа. От него требуется добиться от Заказчика полной информации о решаемой задаче.

2. **Моделирование и формализация задачи**. Цели этого этапа уже обсуждались выше в разделе методики разработки алгоритма. При моделировании важно иметь опыт программирования, знать возможности компьютера и языка программирования и выдвигать гипотезы с учетом этих возможностей. К разработке алгоритма следует приступать только после принятия гипотезы решения задачи.

   Помимо идеи решения задачи, результатами этого этапа должны быть формализованная постановка задачи типа "дано-найти" и достаточное количество контрольных примеров для последующего тестирования программы. К категории "Дано:" обычно относятся данные, вводимые в начале работы программы и обеспечивающие массовость алгоритма. К категории "Найти:" относятся данные, получаемые в результате работы программы.

3. **Разработка алгоритма**. Этот этап представляет собой реализацию идеи решения задачи (см. методику разработки алгоритма).

4. **Тестирование алгоритма**. Этап предполагает проверку алгоритма вручную с использованием подготовленных ранее контрольных примеров. Для сложных задач этот этап может оказаться весьма трудоемким, поэтому опытные программисты пропускают его и тестируют программу.
5. **Программирование алгоритма**. Программирование является формальной записью алгоритма средствами языка программирования.

6. **Тестирование программы**. Тестирование выполняется путем вывода промежуточных результатов работы программы и сравнения их с контрольным примером. Для этого либо используют специальные средства отладки программ, имеющиеся в интегрированной среде языка программирования, либо временно добавляют в программу команды вывода промежуточных значений. Уменьшить трудоемкость поиска ошибок в программе можно более тщательным проектированием алгоритма и планированием процесса тестирования на ранних стадиях разработки программы.

7. **Эксплуатация программы и интерпретация результатов**. В сложных программах может быть недостаточно тестирования для устранения всех ошибок. Очень час-то они обнаруживаются на стадии эксплуатации Заказчиком.

Успех в разработке программы зависит от двух основных факторов: соблюдения описанной выше методики и от опыта программирования. Не следует игнорировать или недооценивать этапы проектирования программы (1 – 5), выполняемые вне компьютера. "Час, потраченный на выбор алгоритма, стоит пяти часов программирования" (Д.Ван-Тассел. Стиль, разработка, эффективность, отладка и испытание программ.- М.: Мир, 1985).

Пример 2

Рассмотрим задачу с достаточно сложным алгоритмом решения для того, чтобы, во-первых, продемонстрировать этапы 1 - 3 рассмотренной методики и, во-вторых - принцип поэтапной детализации алгоритма.

Постановка задачи. Существует способ обойти шахматным конем доску, побывав на каждом поле по одному разу. Построить алгоритм обхода доски.

Идея решения задачи. Очередной ход следует делать на то поле, с которого на другие поля меньше всего ходов.

Формализация задачи. Назовем термином "потенциал поля" количество допустимых ходов коня. Введем следующие обозначения:

- C — матрица 8*8, содержащая потенциалы полей (фрагмент C показан на рис. 13);
- R — матрица 8*8, содержащая решение задачи в виде номеров ходов коня;
- Sx, Sy — массивы из 8 элементов, содержащие смещения коня относительно текущей координаты, необходимые для реализации правила буквы "Г":

Sx = (1, 2, 2, 1,-1,-2,-2,-1);
Sy = (-2,-1, 1, 2, 2, 1,-1,-2).

- x, y — текушие координаты коня;
- x1,y1 — координаты поля с минимальным потенциалом для текущих (x, y);
- m — значение минимального потенциала допустимого поля.

Будем учитывать пройденные поля путем задания соответствующим элементам матрицы C значения 9, т.е. значения вне множества допустимых потенциалов.

1.5 Разработка алгоритма решения задачи

На рис. 8 показан укрупненный алгоритм решения поставленной задачи. На рис. 9 - 12 показаны основные шаги поэтапной детализации основного алгоритма.

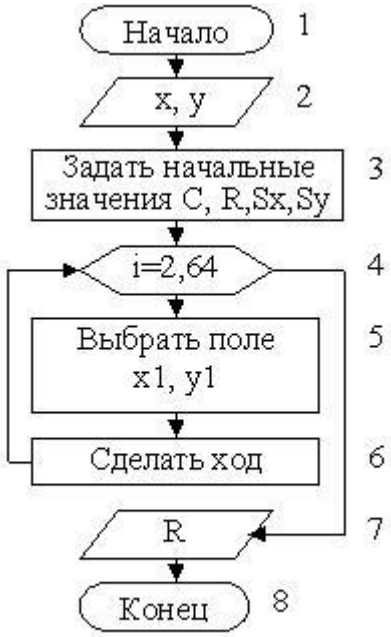


Рис.8.

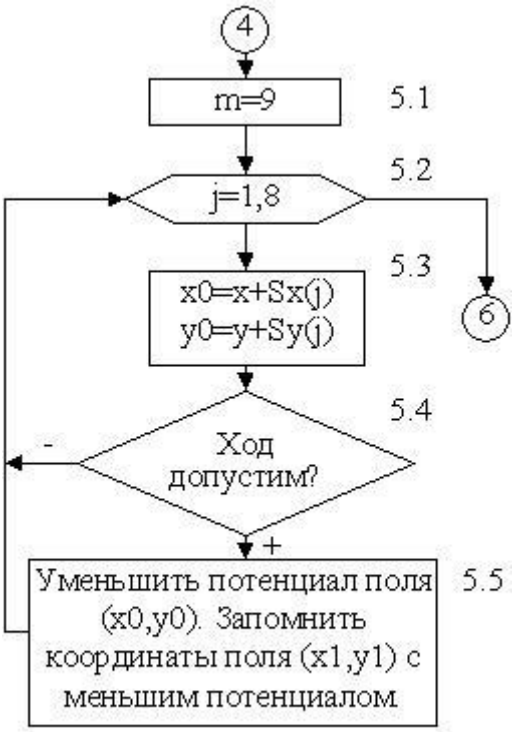


Рис.9.

Следует обратить внимание на нумерацию блоков в детализирующих блок-схемах. Число до первой точки является номером детализируемого блока в основной схеме. Число после первой точки является номером блока в схеме детализации первого уровня и т.д.

Входы в детализирующие блок-схемы и выходы из них показаны окружностями с номерами блоков — источников информации и получателей результатов.

Значком & на рис. 11 обозначена логическая операция И.

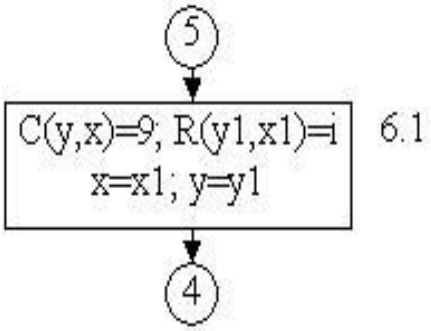


Рис.10.

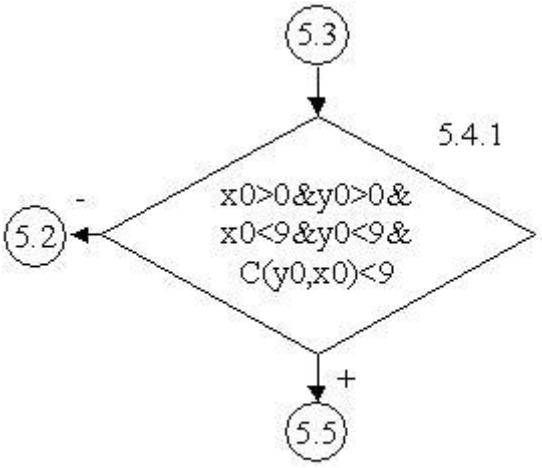


Рис.11

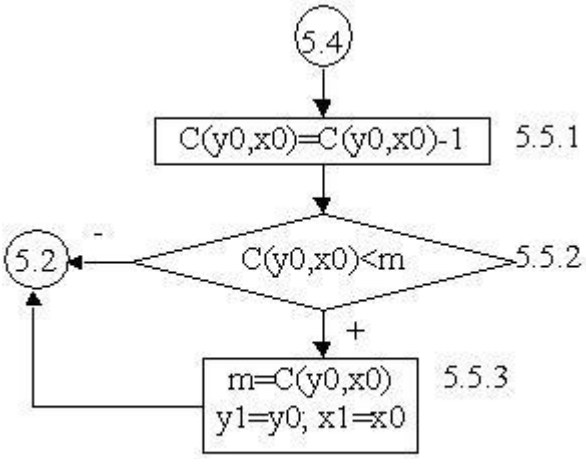


Рис.12

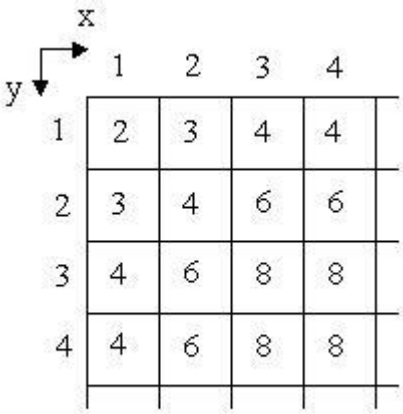
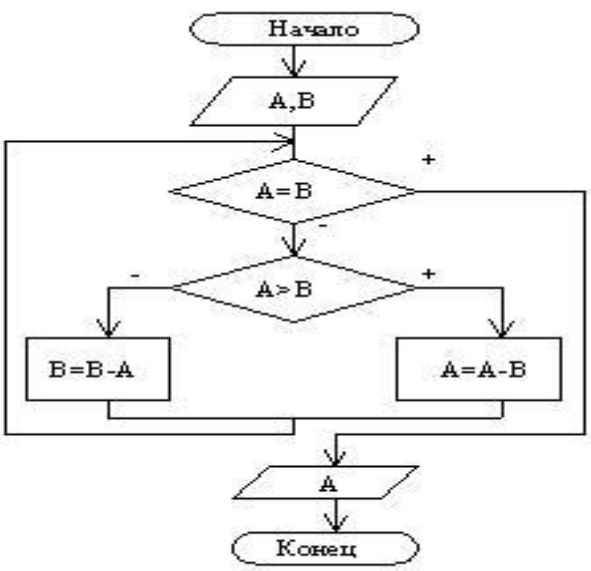


Рис.13

Пример 3

Поскольку тестирование вручную алгоритма решения задачи о шахматном коне было бы достаточно громоздким, рассмотрим технологию тестирования на примере алгоритма Евклида (рис. 14).



A	35			
B	14			
A=B	нет	нет	нет	да
A>B	да	да	нет	
B			7	
A	21	7		
A				7

Рис. 14. Тестирование алгоритма Евклида

Для тестирования вручную следует оставить достаточно свободного места справа от блок-схемы. Контрольный пример не должен быть слишком сложным - это затрудняет тестирование, но и не быть тривиальным - это может привести к случайному совпадению с правильным решением. В первом столбце таблицы справа от блок-схемы, записываются переменные или условия, значения которых могут изменяться. Начиная со второго столбца сверху-вниз записываются результаты выполнения алгоритма. Начало нового цикла соответствует добавлению нового столбца таблицы. Исполнитель (разработчик алгоритма) должен выполнять команды формально, строго придерживаясь предписаний в блок-схеме.

Примеры реализации программ рассмотрены в разделе "Методы процедурного программирования"

Контрольные вопросы

1. Что такое программа?
2. Что понимают под исполнителем?
3. Что представляет собой машинный код?
4. Что такое транслятор? Перечислите типы трансляторов.
5. Как работает интерпретатор? В чем его достоинства?
6. В чем заключается достоинство компиляторов?
7. Какие компоненты необходимы для создания программ? Каково назначение каждого из этих компонентов?
8. Что называется интегрированной системой программирования?

9. Чем характеризуются системы визуального программирования?
10. Какие подходы по способу разработки программ можно выделить? Охарактеризуйте каждый подход.
11. Каковы основные системы программирования?
12. Перечислите основные этапы развития языков программирования. Что понимают под алгоритмом?
13. Каковы способы записи алгоритмов?
14. В чем заключаются основные свойства алгоритма?
15. Перечислите основные алгоритмические структуры и опишите их.
16. Каковы основные принципы разработки алгоритмов?
17. Назовите основные этапы составления алгоритмов.
18. Приведите пример, реализующий этапы алгоритмизации.
19. Каковы основные этапы решения задач с помощью ЭВМ? Дайте характеристику каждому этапу.

2. Введение в программирование на Visual Basic

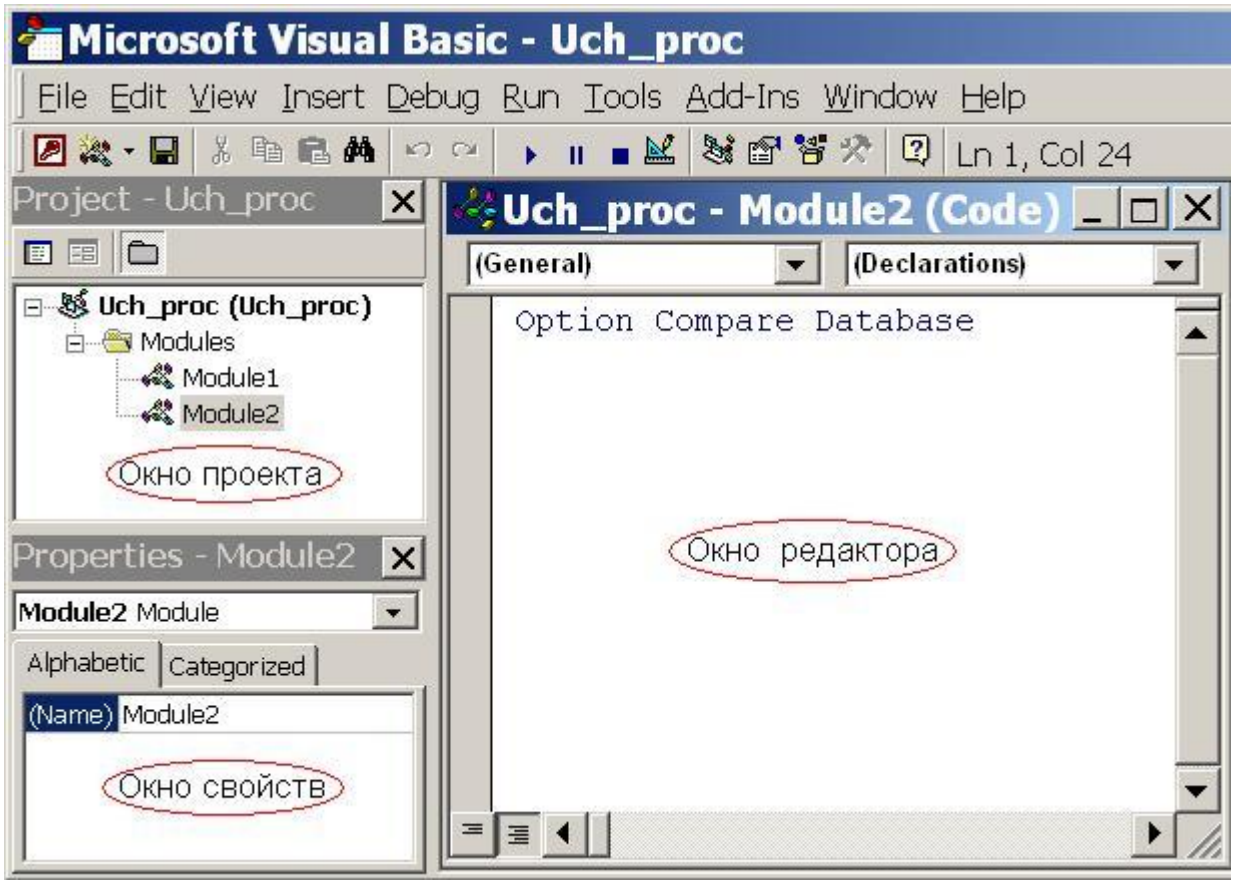
В качестве языка программирования будем использовать Visual Basic, а точнее - Visual Basic for Applications (VBA), встроенный в среду MS Access. Причинами этого выбора являются:

- для использования языка не требуется установки компилятора на компьютере пользователя – VBA встроен в среду MS Access;
- в VBA реализованы основные концепции процедурного и объектно-ориентированного программирования;
- VBA для MS Access является мощным инструментом без которого трудно обойтись при разработке сложных СУБД.

Следует также заметить, что в рамках курса информатики мы рассматриваем лишь основы программирования на Visual Basic. На этом уровне изучения приводимые коды программ не требуют никакой коррекции при их использовании в среде универсальной версии Visual Basic, например, VB 6.0.

2.1 Среда разработки программ

Для открытия окна интегрированной среды разработки программ в MS Access необходимо в окне БД выбрать пункт ***Модули*** и нажать кнопку ***Создать***. Фрагмент окна интегрированной среды показан на рисунке.



Основными пунктами меню, необходимыми для запуска и отладки программы являются:

- ***Run/Run Sub*** – запуск программы;
- ***Run/Break*** – приостановка выполнения программы;
- ***Run/Reset*** – прерывание выполнения программы;
- ***Debug/Step Into*** – выполнить очередной оператор программы;
- ***Debug/Step Over*** – выполнить оператор программы без входа в процедуры;
- ***Debug/Toggle BreakPoint*** – установить точку прерывания;
- ***Debug/Clear All BreakPoints*** – очистить все точки прерывания;

Текст (код) программы набирают в окне редактора. После набора программу следует запустить на выполнение командой ***Run/Run Sub***. Если в программе обнаружены ошибки, то используются команды отладки, перечисленные выше.>

2.2 Структура программного кода

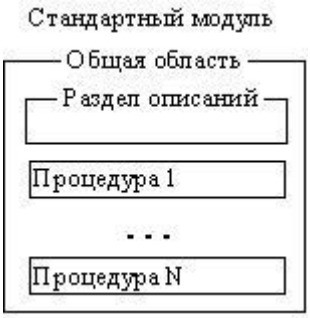
Основой программ на VBA являются процедуры и функции, состоящие из инструкций, которые выполняют необходимые действия и вычисления. Каждая процедура имеет имя, по которому она вызывается на выполнение.

Порядок записи процедур и функций безразличен, однако должен существовать главный объект, с которого начинается выполнение программы. В большинстве случаев таким объектом является процедура Main().

Программы на языке Visual Basic в среде Access хранятся в модулях. **Модуль** является структурой, сохраняющей некоторый набор описаний и процедур, или способом организации процедур.

Модули подразделяются на стандартные и модули форм и отчетов.

Стандартные модули. Стандартные модули являются отдельными объектами Access и представлены в окне БД. В них хранятся процедуры, доступные из любых других объектов базы данных. Вызов этих процедур может осуществляться из процедур обработки событий, процедур других стандартных модулей, макросов и просто из выражений.



Модули форм и отчетов. Любая форма и любой отчет базы данных содержит встроенный модуль. Модуль формы или отчета создается автоматически при создании формы или отчета и является частью их описания. Модуль существует пока существует форма или отчет, копируется и удаляется вместе с ними. Процедуры модулей связываются с событиями этих объектов.

Процедуры и функции

Процедуры имеют следующий синтаксис:

[Private | Public] Sub <Имя>(<Формальные аргументы>)
<Тело процедуры>
End Sub

Здесь и далее угловые скобки (< и >) содержат пояснения, на место которых должны быть подставлены реальные текстовые конструкции, соответствующие синтаксическим правилам языка. Квадратные скобки означают необязательность применения записанных в них служебных слов. Вертикальная черта означает возможность выбора одного из служебных слов.

Вызов процедуры общего назначения выполняется по имени:

<Имя>(<Фактические аргументы>)

При вызове процедуры фактические аргументы подставляются на место формальных и управление выполнением передается процедуре. Аргументы могут быть входными, выходными или модифицируемыми. Через входные аргументы процедура получает данные при обращении к ней. Выходные аргументы возвращают результаты выполнения процедуры. Модифицируемые аргументы являются одновременно входными и выходными.

Функция общего назначения построена также, как процедура, однако, результат работы функции передается (возвращается) через ее имя. Поэтому, как и в математике, обращения к функциям можно использовать внутри арифметических и логических выражений. Синтаксис функции:

[Private | Public] Function <Имя>(<Аргументы>)<Описание функции>
<Тело функции>
End Function

Для того, чтобы функция возвращала результат через имя, в теле функции должна присутствовать хотя бы одна команда присваивания типа

<Имя>=<выражение>

Служебные слова Private и Public задают область видимости процедур и функций. Private делает объект доступным только внутри данного модуля. Public делает объект дос-тупным из другого модуля.

2.3 Описание переменных

Модуль, тело процедуры или функции обычно начинаются с раздела описаний. Он содержит определения **переменных** и **констант**, которые используются в модуле и процедурах. С помощью переменных в процедуры передаются аргументы, в ходе выполнения процедур сохраняются рабочие промежуточные значения, осуществляется обмен данными между процедурами. Переменные существуют только внутри модулей, процедур или функций. Каждая переменная имеет имя.

Основной инструкцией для явного описания переменных является инструкция Dim. При размещении инструкции описания в разделе описаний модуля создается переменная, которая может использоваться внутри модуля. При размещении инструкции описания внутри процедуры создается переменная, которая может использоваться только внутри процедуры. При определении переменной для нее указывается тип данных.

Формат инструкции описания переменной:

Dim <Имя переменной> [As <Тип данных>]

Следующая инструкция создает переменную X и указывает для нее текстовый (строковый) тип данных String:

Dim X As String

Если разместить данную инструкцию внутри процедуры, то переменная X может быть использована только внутри этой процедуры. Если поместить данную инструкцию в раздел описаний модуля, то переменная X будет доступна для любых процедур в данном модуле, но недоступна для процедур в других модулях. Для того чтобы сделать данную переменную доступной для всех процедур в базе данных, следует описать ее как общую с помощью инструкции **Public**:

Public X As String

В языке Visual Basic действуют следующие соглашения на имена процедур, переменных и констант:

- должны начинаться с буквы;
- могут включать буквы, цифры и символы подчеркивания;

- не должны включать знаки препинания или пробелы;
- не должны совпадать с ключевыми словами языка Visual Basic.

Основными типами данных, используемыми при описании переменных, являются:

- **Integer** – целое число (2 байта);
- **Long** – длинное целое число (4 байта);
- **Single** – десятичное число одинарной точности (4 байта);
- **Double** – десятичное число двойной точности (8 байтов);
- **Currency** – десятичное число с фиксированной точкой (8 байтов);
- **String** – строка текста (до 65400 символов);
- **Byte** – целое от 0 до 255 (1 байт);
- **Boolean** – логическое значение True или False (2 байта);
- **Date** – дата и время (8 байтов);
- **Object** – экземпляр класса (4 байта);
- **Variant** – любой из перечисленных выше типов (16 байтов + 1 байт/символ).

Указание типа данных в инструкции описания не является обязательным. Если тип данных не указан, по умолчанию переменная получит тип Variant.

Данные этого типа интерпретируются в Visual Basic в зависимости от операции, в которой они используются, т.е. их тип может меняться. При этом производится преобразование одного типа данных в другой. Использование типа данных Variant вместо любого типа данных обеспечивает большую гибкость при обработке. Однако это порождает некоторые трудности и может приводить к возникновению ошибок.

Допускается описание нескольких переменных в одной инструкции. Если требуется явно описать тип данных, то необходимо включить описание типа данных для каждой переменной. В следующей инструкции переменные X, Y и Z описываются с типом данных **Integer**:

Dim X As Integer, Y As Integer, Z As Integer

В следующей инструкции переменные X и Y описываются с типом данных Variant; а переменная Z с типом данных Integer:

Dim X, Y, Z As Integer

Для того чтобы запретить неявное описание переменных в Visual Basic, следует поместить в раздел описания модуля инструкцию **Option Explicit**. Эта инструкция требует явного описания всех переменных, используемых в модуле. Если в модуль включена инструкция **Option Explicit**, то Visual Basic генерирует ошибку при компиляции, если обнаруживается имя переменной, которая не была описана ранее.

3. Операторы управления выполнением программы

3.1 Оператор присваивания

Присваивание значений переменным осуществляется с помощью **оператора присваивания**. В этом операторе слева стоит имя переменной, а справа присваиваемое значение или выражение. Например:

```
X=200*0.8/70
Y="Петров Семен Иванович"
Z=X/80+30
Y=Forms![СТУДЕНТ]![ФИО]
```

Оператор ветвления

Оператор ветвления предназначен для изменения порядка выполнения программы по некоторому условию. Синтаксис оператора:

```
If <условие1> Then
<Блок 1>
[ ElseIf <условие2> Then
<Блок 2>
· · ·]
[Else
<Блок N>]
End If
```

Оператор работает следующим образом: если (**If**) выполняется **<условие 1>**, то (**Then**) будет выполнена последовательность операторов **<Блок 1>**, иначе, если (**ElseIf**) выполняется **<условие 2>**, то будет выполнен **<Блок 2>** и т.д., иначе (**Else**) - **<Блок N>** .

Допустима форма записи оператора ветвления в одну строку:

If <условие1> Then <Оператор 1>: <Оператор 2>: . . . : <Оператор N>

В этом случае не указывается конец оператора (**End If**) как в первом варианте.

Оператор цикла For . . . Next

Синтаксис:

For <i>=<i0> To <iN> [Step <ih>]
<Блок>
Next [<i>]

Значения <i>, <i0>, <iN>, <ih> соответственно означают:

- <i> - параметр цикла;
- <i0>, <iN> - начальное и конечное значения параметра цикла;
- <ih> - шаг изменения параметра цикла.

Обычно перечисленные величины являются целыми числовыми. Если шаг не указан, то он по умолчанию принимается равным 1.

Для досрочного выхода из цикла используется команда **Exit For**.

Цикл Do с предусловием

Синтаксис:

Do [While|Until] <условие>
<Блок>
Loop

Если выбрано служебное слово **While**, то цикл продолжается, пока выполняется условие; если **Until** – то прекращается, когда условие выполняется.

Цикл Do с постусловием

Синтаксис:

Do

<Блок>

Loop [{While|Until}] <условие>

Значение служебных слов цикла аналогично циклу с предусловием, но здесь сначала выполняются действия в блоке, а затем проверка выхода из цикла.

В обоих циклах Do для досрочного выхода из цикла используется команда **Exit Do**.

3.2 Операторы ввода-вывода

Одним из важнейших свойств программы является **массовость**. Она обеспечивается возможностью ввода различных исходных данных в программу и получения различных результатов без изменения кода самой программы. Кроме того, пользователю необходимо наблюдать результаты работы программы, что обеспечивается операторами вывода данных.

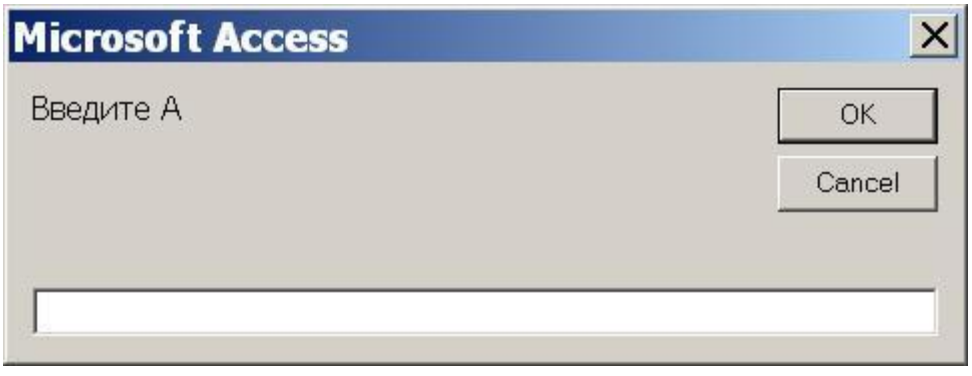
В Visual Basic возможны несколько способов ввода данных в программу. Одним из самых простых является использование функции InputBox, имеющей следующий синтаксис:

InputBox("<Текст подсказки>")

Например, при выполнении команды присваивания

S = InputBox("Введите A")

будет выведено стандартное окно:



После ввода значения в текстовое поле и нажатия кнопки Ok, переменной S будет присвоен введенный текст.

Следует иметь в виду, что функция InputBox() возвращает текстовое значение. Поэтому при необходимости ввести число, следует преобразовать возвращаемое значение в требуемый числовой тип. Для преобразования в основные числовые типы используются функции:

- **CInt()** – для преобразования в целый тип Integer;
- **CSng()** – для преобразования в вещественный тип Single.

Таким образом, для ввода целого значения N следует записать:

N = CInt(InputBox("Введите N"))

При преобразовании текстовых значений в вещественные с помощью функции CSng(), в окне InputBox в качестве разделителя целой и десятичной частей числа должна быть введена запятая (в выражениях разделителем является точка!). Если введен текст, не соответствующий образу числа, то функции CInt() и CSng() выдают сообщение об ошибке.

Для вывода результатов вычислений можно использовать процедуру MsgBox():

MsgBox (<текст>)

Выводимое значение должно иметь текстовый тип. Для преобразования числа любо-го типа в текст используется функция

CStr(<числовая переменная>)

Если в одном окне MsgBox требуется вывести несколько чисел, каждое из них сле-дует преобразовать в текстовый тип и "склеить" оператором +, например, при выполнении фрагмента программы:

```
N = 5
Pi = 3.14
MsgBox(CStr(N)+" "+CStr(Pi))
```

будет выведено следующее окно:

Обратите внимание, что функция CStr(Pi) возвратила значение, разделенное запятой, несмотря на то, что в программе разделителем в числе 3.14 является точка.

Контрольные вопросы

1. Почему в качестве языка программирования выбран язык Visual Basic for Applications (VBA)?
2. Как открыть окно интегрированной среды разработки программ VBA?
3. Каковы основные элементы интегрированной среды?
4. Охарактеризуйте основные пункты меню интегрированной среды разработки программ?
5. Что является основой программ на VBA?
6. Что такое модуль?
7. Как подразделяются модули в VBA?
8. Что представляют собой стандартные модули? Как они создаются?
9. Как создаются модули форм и отчетов?
10. Что такое процедура? Каков синтаксис процедур?
11. Как осуществляется вызов процедур? Что при этом происходит?
12. Как осуществляется обмен данными между процедурами и программой?
13. Какие аргументы используются при описании процедур?
14. Каков синтаксис функции? В чем ее отличие от процедуры?
15. Каково назначение раздела описаний?
16. Как используются переменные в процедуре?
17. Каков формат инструкции описания переменных?
18. Какова область видимости переменных, описанных в модуле? в процедуре или функции?

19. Как описать общую переменную, доступную для всех процедур в базе данных?
20. По каким правилам можно задать имя переменной в языке VB?
21. Перечислите и охарактеризуйте основные типы данных в VB?
22. Каковы особенности типа Variant?
23. Как запретить неявное описание переменных в VB?
24. Перечислите и охарактеризуйте операторы управления выполнением программы. Приведите примеры их использования.
25. Как можно ввести данные в программу?
26. С помощью каких функций преобразования можно преобразовать строковое значение в число?
27. Как можно вывести на экран результат вычислений?

3.3 Методы программирования с использованием типовых алгоритмических конструкций

Рассмотрим примеры разработки программ, основанные на методике компьютерного решения задач (раздел "Основы алгоритмизации и программирования"), реализуемые средствами языка Visual Basic (раздел "Введение в программирование на Visual Basic").

Задача 1

Дан кирпич прямоугольной формы со сторонами A, B, C и прямоугольное отверстие в стене со сторонами X и Y. Определить, пройдет ли кирпич в отверстие, если допускается располагать его грани только параллельно сторонам отверстия.

Решение

Основная идея решения заключается в упорядочивании сторон кирпича и отверстия по возрастанию. Если меньшая сторона кирпича меньше меньшей стороны отверстия, а вторая по размеру сторона кирпича меньше большей стороны отверстия, то кирпич пройдет, иначе — нет. Таким образом, требуется решить подзадачи упорядочивания сторон кирпича и сторон отверстия. Поскольку цель задачи — дать ответ "пройдет" или "не пройдет", будем выполнять упорядочивание, не вводя новых переменных. Для сторон отверстия задача решается просто: если X>Y, то обменять значения X и Y. Упорядочить стороны кирпича

можно за три обмена: A и B, B и C и вновь A и B. Блок-схема алгоритма решения задачи показана на рис. 1.

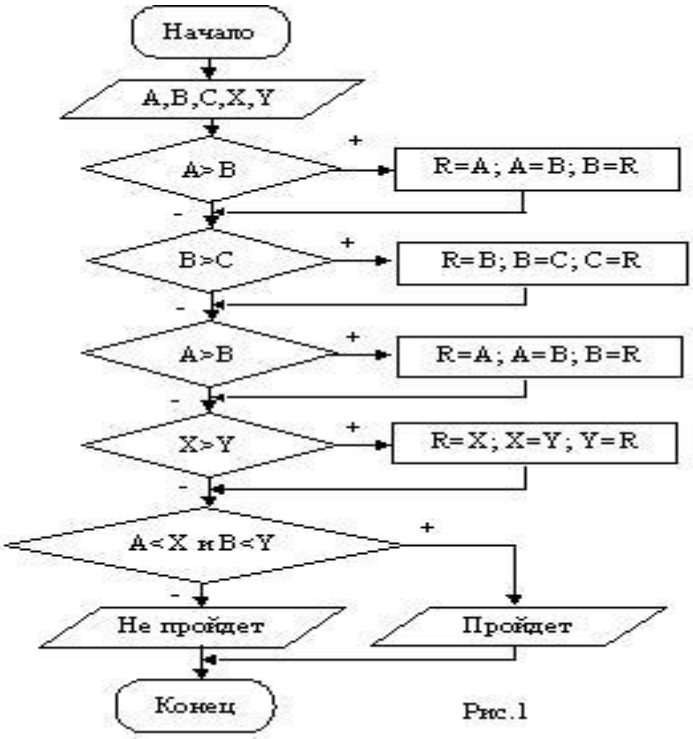


Рис.1

Код программы:

```
Option Explicit
Private Sub Main()
  Dim A As Single, B As Single
  Dim C As Single, X As Single
  Dim Y As Single, R As Single
  A = CSng(InputBox("Введите A"))
  B = CSng(InputBox("Введите B"))
  C = CSng(InputBox("Введите C"))
  X = CSng(InputBox("Введите X"))
  Y = CSng(InputBox("Введите Y"))
  If A>B Then R=A: A=B: B=R
  If B>C Then R=B: B=C: C=R
  If A>B Then R=A: A=B: B=R
  If X>Y Then R=X: X=Y: Y=R
  If A<X AND B<Y then
    MsgBox("Пройдет")
```

```
  Else
    MsgBox("Не пройдет")
  End If
End Sub
```

В коде программы следует обратить внимание на форму записи команд ветвления, используемых для обмена величин: каждая из них занимает одну строку. При этом команды присваивания после служебного слова Then разделены двоеточием. Для такой синтаксической формы не следует использовать команду End If.

С алгоритмической точки зрения в задаче 1 применены следующие новые приемы:

- обмен значений двух переменных с использованием третьей переменной;
- применение простых условий и однотипных команд ветвления для упорядочивания величин, что снижает вероятность ошибки в программе.

Плохим стилем программирования считается применение в задаче 1 сложных условий.

Задача 2

Вычислить сумму $1-\frac{x}{2!}+\frac{x^2}{3!}-\frac{x^3}{4!}+\ldots+(-1)^{k-1}\frac{x^{k-1}}{n!}$

Решение

Моделирование задачи 2 показывает, что для вычисления суммы можно использовать цикл For-Next, поскольку заранее известно количество слагаемых. Кроме того, не следует вычислять заново факториал для очередного слагаемого: гораздо проще вычислить факториал, опираясь на значение факториала для предыдущего слагаемого. Аналогично организуется чередование знаков слагаемых: введем целую переменную Z=1 и будем в цикле выполнять команду Z=-Z.

Для решения задачи введем следующие величины:

- S — искомая сумма;
- F — значение факториала;
- C — значение числителя;

- Z — знак слагаемого (+1 или –1).

Таким образом, в задаче 2 дано: N, X; надо получить S.

Контрольный пример: при N=4 и X=3 получим S = -0.125.

Алгоритм решения задачи показан на рис. 2.

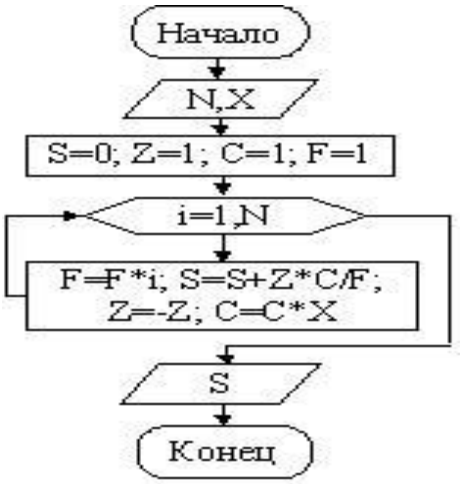


Рис.2.

Код программы:

```
Option Explicit
Private Sub Main()
  Dim S As Single, C As Single
  Dim X As Single, Z As Integer
  Dim i As Integer, N As Integer, F As Long
  N=CInt(InputBox("Введите N"))
  X=CSng(InputBox("Введите X"))
  S=0: Z=1: C=1: F=1
  For i=1 To N
    F=F*i
    S=S+Z*C/F
    Z=-Z: C=C*X
  Next
  MsgBox (CStr(S))
End Sub
```

Обратите внимание, что порядок расположения команд внутри цикла не безразличен. Например, перенос вычисления F в конец цикла приведет к ошибочному результату. Обычно, при разработке алгоритма решения задачи он записывается не последовательно от начала к концу, а от главного блока к началу и к концу. В рассмотренном примере главным является блок внутри цикла.

Основным методом, примененным в задаче 2, является вычисление отдельных частей очередного слагаемого с использованием их значений для предыдущего слагаемого.

Задача 3

Сколько членов последовательности $1, \frac{1}{2}, \frac{1}{3}, \ldots$ надо просуммировать для того, чтобы сумма превысила значение S?

Решение

Основной особенностью задачи 3 является то, что количество повторений N для вычисления суммы заранее неизвестно. В определенном смысле задача 3 является обратной к задаче 2: дано S, требуется найти N. Поэтому применять цикл For-Next нельзя. Для подготовки контрольного примера составим следующую таблицу:

Номер цикла i	**1**	**2**	**3**	**4**	**5**
Сумма S1	1	1,5	1.833333	2.083333	2.283333

Из таблицы видно, что при S=2.09 решением является N=5, а при S=2.08 получим N=4. Выбранные значения S удобны тем, что при небольшом изменении S от 2.08 до 2.09 изменяется решение задачи. Если тестирование алгоритма при этих исходных данных даст положительный результат, можно с большой вероятностью утверждать, что алгоритм не имеет ошибок.

Совет. *Контрольные примеры для вычислительных задач следует подбирать на основе предварительного численного эксперимента. Контрольный пример не должен быть тривиальным — тогда при неправильном алгоритме можно получить правильное решение.*

С другой стороны, он не должен быть слишком сложным, чтобы можно было протестировать алгоритм вручную.

Идеей решения задачи 3 является накопление суммы S1 в цикле до тех пор, пока она не превышает заданной границы S.

Блок-схема алгоритма решения задачи показана на рис. 3.

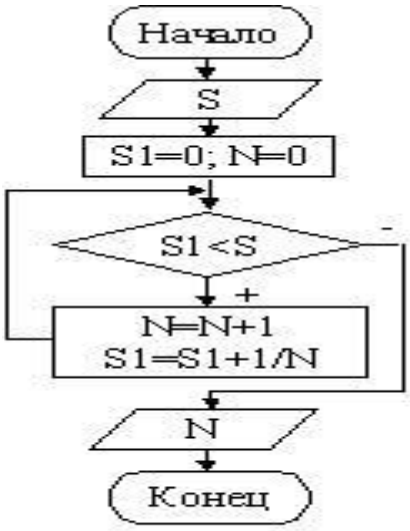


Рис.3.

Код программы:

```
Option Explicit
Private Sub Main()
  Dim S As Single, S1 As Single
  Dim N As Integer
  S=CSng(InputBox("Введите S:"))
  S1=0: N=0
  Do While S1<S
    N=N+1
    S1=S1+1/N
  Loop
  MsgBox("N="+Cstr(N))
End Sub
```

Основным приемом, примененным при решении задачи 3, является использование цикла типа "пока" при неизвестном количестве повторений.

Задача 4

Последовательность значений X_i, Y_i вычисляется по формулам:

$$X_i = 0.3\ X_{i-1} + 0.4\ X_{i-2} + 0.2\ Y_{i-1};$$
$$Y_i = 0.2\ Y_{i-1} + 0.3\ Y_{i-2} + 0.4\ X_{i-1}; \qquad i = 2, 3, \ldots$$

Вычислить $\sum_{i=0}^{N} (X_i + Y_i)$ при $X_0=X_1=1$ и $Y_0=Y_1=2$.

Решение

В данной задаче не следует применять массивы — лучше использовать простые переменные. Поэтому для численного эксперимента введем следующие обозначения: X2, X1, X0, Y2, Y1, Y0 – соответственно для X_i, X_{i-1}, X_{i-2}, Y_i, Y_{i-1}, Y_{i-2}. Результаты численного эксперимента показаны в таблице.

i	**0**	**1**	**2**	**3**	**4**
X0			1	1	1,1
X1		1	1	1,1	1,01
X2	1	1	1,1	1,01	1,007
Y0			2	2	1,4
Y1		2	2	1,4	1,32
Y2	2	2	1,4	1,32	0,987
Сумма	3	6	8,5	10,83	12,925

Таким образом, в задаче 4 в качестве контрольного примера можно использовать следующий: при N=4 должны получить сумму S=12.925.

Из таблицы видно, что после вычисления очередных значений X2 и Y2 и добавления их к сумме, необходимо изменить значения X1, Y1, X0, Y0 для подготовки очередного цикла. При этом следует начинать с X0 и Y0: X0=X1; Y0=Y1, а затем изменить X1 и Y1: X1=X2:

Y1=Y2. Следует также учесть, что для вычисления суммы может быть задано любое целое N>=0, но применение формул начинается только с N=2. Блок-схема алгоритма решения показана на рис. 4.

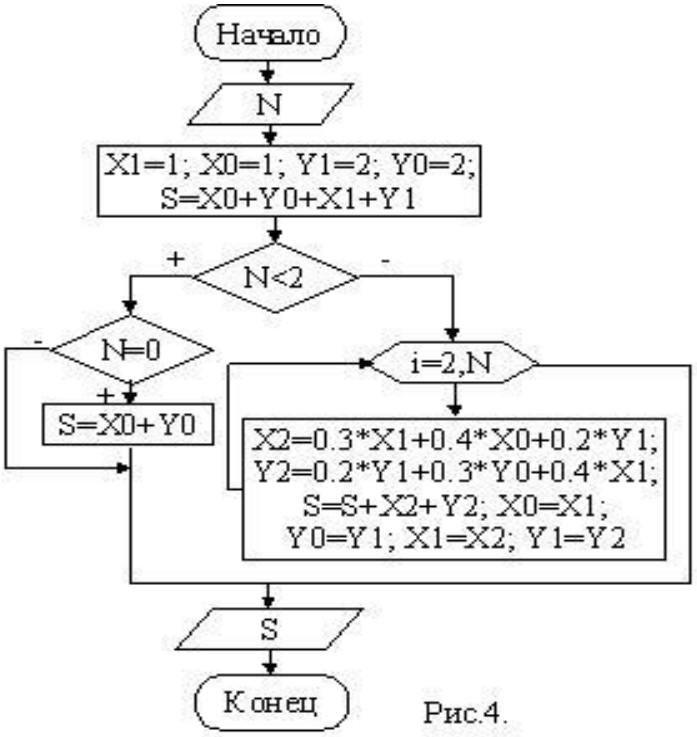


Рис.4.

Код программы

```
Option Explicit
Private Sub Main()
  Dim S As Single, X0 As Single
  Dim X1 As Single, X2 As Single
  Dim Y0 As Single, Y1 As Single
  Dim Y2 As Single, N As Integer
  Dim i As Integer
  N=CInt(InputBox("Введите N:"))
  X1=1: X0=1: Y1=2: Y0=2
  S=X0+Y0+X1+Y1
  If N<2 Then
    If N=0 Then S=X0+Y0
    Else
    For i=2 To N
      X2=0.3*X1+0.4*X0+0.2*Y1
```

```
    Y2=0.2*Y1+0.3*Y0+0.4*X1
    S=S+X2+Y2
    X0=X1: Y0=Y1: X1=X2: Y1=Y2
  Next
 End If
 MsgBox("S="+CStr(S))
End Sub
```

В решении задачи 4 применены два новых приема:

- использованы простые переменные (но не массивы) для вычисления элементов последовательности, поскольку очередные значения X и Y можно вычислить, опираясь на только два предыдущих значения;
- для подготовки к очередному циклу применено обновление переменных, начиная с "самых старых" значений.

Контрольные вопросы

1. Какова основная идея решения задачи 1 (о кирпиче и прямоугольном отверстии)?
2. Какие новые алгоритмические приемы применены при решении задачи 1?
3. Каков основной метод, примененный в задаче 2?
4. Как правильно подбирать контрольные примеры для вычислительных задач?
5. Почему при решении некоторых задач, требующих вычисления элементов последовательности, можно использовать простые переменные для обозначения элементов последовательности, а не массивы?

4. Основы объектно-ориентированного программирования

Концепция ООП возникла в середине 80-х годов. Главная ее идея в том, что программное приложение, как и окружающий нас мир, должно состоять из объектов, обладающих собственными свойствами и поведением. ООП объединяет исполняемый код программы и ее данные в объекты, что упрощает создание сложных программных приложений. Например,

можно организовать коллективную работу над проектом, где каждый участник создает собственный класс объектов, который становится доступным другим участникам проекта.

При объектно-ориентированном подходе программные задачи распределяются между объектами программы. Объекты обладают определенным набором свойств, методов и способностью реагировать на события (нажатие кнопок мыши, интервалы времени и т.д.). В отличие от процедурного программирования, где порядок выполнения операторов программы определяется порядком их следования и командами управления, в ООП порядок выполнения процедур и функций определяется, прежде всего, событиями.

Чтобы проект можно было считать объектно-ориентированным, объекты должны удовлетворять некоторым требованиям. Этими требованиями являются **инкапсуляция**, **наследование** и **полиморфизм**.

- **Инкапсуляция** — означает, что объекты скрывают детали своей работы. Инкапсуляция позволяет разработчику объекта изменять внутренние принципы его функционирования, не оказывая никакого влияния на пользователя объекта. В VB этот принцип реализуется, в основном, за счет применения описаний Private и Public.
- **Наследование** — означает, что новый объект можно определить на основе уже существующих объектов, при этом он будет содержать все свойства и методы родительского. Наследование полезно, когда требуется создать новый объект, обладающий дополнительными свойствами по сравнению со старым. Следует заметить, что VB не поддерживает наследования в строгом смысле этого понятия.
- **Полиморфизм** — многие объекты могут иметь одноименные методы, которые могут выполнять разные действия для разных объектов. Например, оператор "+" для числовых величин выполняет сложение, а для текстовых — склеивание.

В ООП центральным является понятие **класса**. **Класс** – это шаблон, по которому создаются объекты определенного типа. **Класс** объединяет в себе **данные** и **методы их обработки**.

Объекты — это экземпляры определенного класса. Например, кнопки или текстовые поля, устанавливаемые на форме являются экземплярами соответствующих стандартных классов VB.

Создать собственный класс в VB можно с помощью модуля класса. **Модуль класса** состоит из элементов трех типов: **свойств**, **методов**, **событий**, оформляемых в виде описаний, процедур и функций внутри контейнера — модуля. У модуля класса нет собственного

пользовательского интерфейса — формы, однако он может использовать контейнер формы, для чего должен содержать соответствующие методы.

Элементы управления — это объекты, используемые при разработке пользовательского интерфейса.

4.1 Технология визуального программирования в среде VBA

Основой для визуального программирования являются формы. Программирование в среде Access начинается с создания формы в режиме конструктора. В этом случае открывается доступ к панели элементов, показанной на рисунке.

Программисту необходимо установить на форме требуемый элемент управления нажатием соответствующей кнопки на панели элементов и "протяжкой" указателем мыши в нужном месте формы.

Для того, чтобы обеспечить выполнение элементом управления требуемых действий, его необходимо запрограммировать. Это можно сделать, вызвав через контекстное меню окно свойств элемента управления. На вкладке ***События*** следует выбрать подходящее событие и нажать кнопку с тремя точками в выбранной строке. В результате откроется окно редактора VBA с заготовкой процедуры обработки события. Программа управления вписывается внутрь этой заготовки.

Например, для наиболее часто используемого элемента управления – кнопки основным событием является нажатие кнопки. Процедура обработки этого события может содержать любой код, например, обращение к другой процедуре, выполняющей обработку данных в таблице.

Кроме кнопки очень часто используются элементы управления "Надпись" и "Поле".

Надпись применяется для вывода текста, который не должен изменяться пользователем (например, заголовок какого-либо объекта управления).

Поле используется для ввода, вывода и редактирования пользователем текстовой (символьной) информации.

Рассмотрим технологию программирования с использованием элементов управления на примере. При этом будем придерживаться общей методики программирования, рассмотренной в предыдущих разделах.

1. Постановка задачи.

В последовательности из N чисел найти непрерывную подпоследовательность, имеющую максимальную сумму. Интерфейс пользователя должен иметь вид, показанный на рисунке.

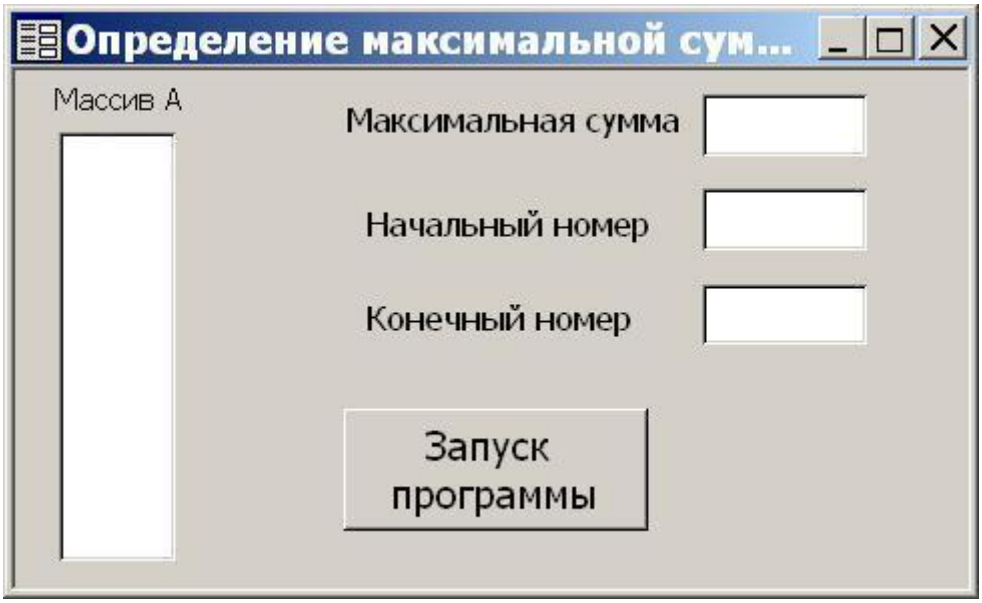


Рис. 1. Вид пользовательского интерфейса

2. Формализация задачи.

Для рассматриваемой задачи введем обозначения:

- A — массив, содержащий исходную последовательность чисел;
- MaxSum — максимальная сумма;
- NumBegin, NumEnd — номера первого и последнего элементов подпоследовательности, образовавшей максимальную сумму.

Массив — это множество однотипных переменных, имеющих одно имя, доступ к которым определяется индексом. В Visual Basic индексы заключаются в круглые скобки после имени массива, например ***A(5)***. При описании массива следует указывать его верхнюю границу,

например, описание ***Dim A(100) As Single*** определяет массив A из 101 элемента вещественного типа. Такое количество элементов обусловлено тем, что нижняя граница индексов любых массивов равна нулю. Следует заметить, что описание только резервирует необходимый объем памяти для хранения массива. Количество элементов массива, используемых при решении задачи может быть любым в пределах описанных границ.

Таким образом, формальной постановкой решаемой задачи является:

- дано: N, A;
- найти: MaxSum, NumBegin, NumEnd.

3. Моделирование задачи и подготовка контрольного примера.

Для сформулированной задачи имеется несколько решений. Оптимальное заключается в использовании однократного просмотра всех N элементов и базируется на том, что при получении отрицательной суммы, вся подпоследовательность, давшая эту сумму, может быть исключена из дальнейшего рассмотрения.

Контрольный пример: A = (2.5; -3.1; 1.8; 4.2; 2.9; -1.1; 2.9; 2.3; -7.9). Максимальную сумму в этом примере, равную 13, образуют элементы с 3-го по 8-й.

4. Разработка алгоритма решения задачи.

Алгоритм для рассматриваемого примера и его детализация показаны на рис. 2-4.

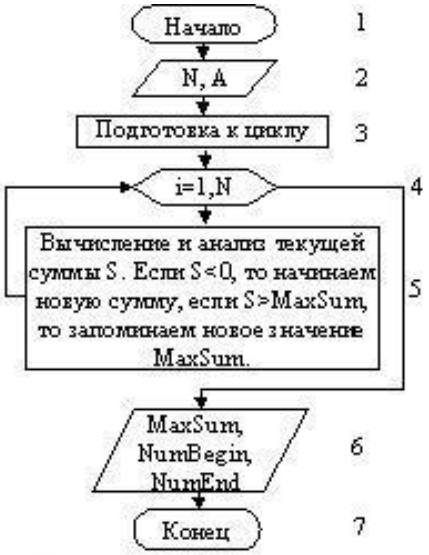


Рис. 2. Основная схема алгоритма

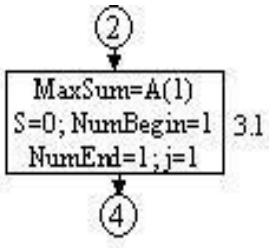


Рис. 3. Детализация блока 3

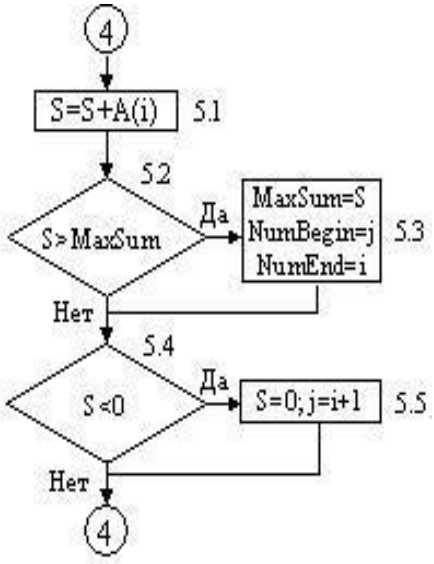


Рис. 4. Детализация блока 5

5. Разработка интерфейса

Для хранения исходных данных создадим таблицу **Последовательность** с одним числовым полем (одинарное с плавающей точкой) под именем **Элемент**. Заполним таблицу **Последовательность** значениями, подготовленными в качестве контрольного примера.

В режиме конструктора создадим форму ***"Определение максимальной суммы"***. Для того, чтобы убрать с формы лишние детали, зададим ее свойства, как показано на рисунке.

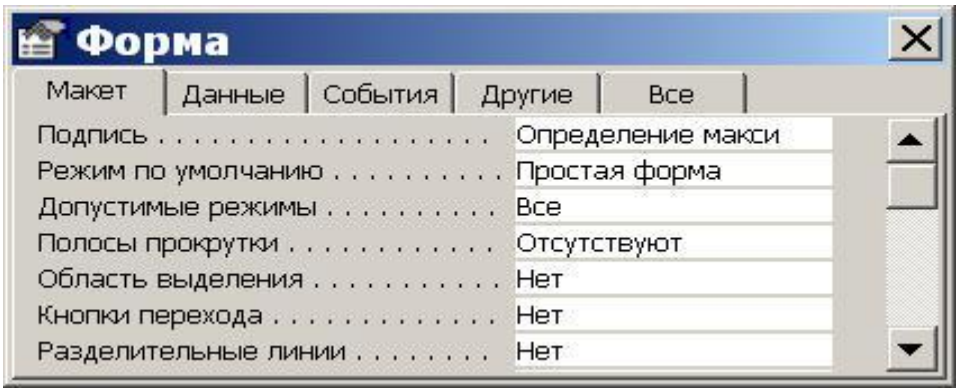


Рис. 5. Свойства формы

Для вывода массива A установим на форме элемент ***Список***, как показано на рис. 1. Если на панели элементов нажата ***кнопка Мастера***, то мастер потребует определить некоторые свойства списка. Главным из них является свойство ***Источник записей***, где следует указать таблицу ***Последовательность***.

Вывод максимальной суммы, начального и конечного номеров будем выполнять в простые поля. Следует проверить, чтобы имена полей были соответственно ***Поле1, Поле3*** и ***Поле5***, а командной кнопки — ***Кнопка0***. В этом случае имена будут совпадать с приводимыми ниже в листинге программы

6. Программирование алгоритма

Для записи программы вызовем контекстное меню кнопки запуска вычислений (рис.1) и выберем пункт ***Обработка событий...*** . В окне ***Построитель*** следует выбрать ***Программы***. Откроется ***окно редактора*** VBA с заготовкой:

Private Sub Кнопка0_Click()

End Sub

Код обработки нажатия кнопки следует вписать внутрь этой процедуры. Ниже при-веден полный листинг программы с комментариями к командам программы.

Работа программы может зависеть от настройки интегрированной среды. Поэтому, находясь в среде редактора программы, следует обратиться к меню ***Tools/References...*** и подключить библиотеки, как показано на рис. 6. Важен не только набор библиотек, но и их расположение в окне ***References***: верхние имеют приоритет по сравнению с нижними. Для изменения приоритета в окне имеются две кнопки ***Priority***.

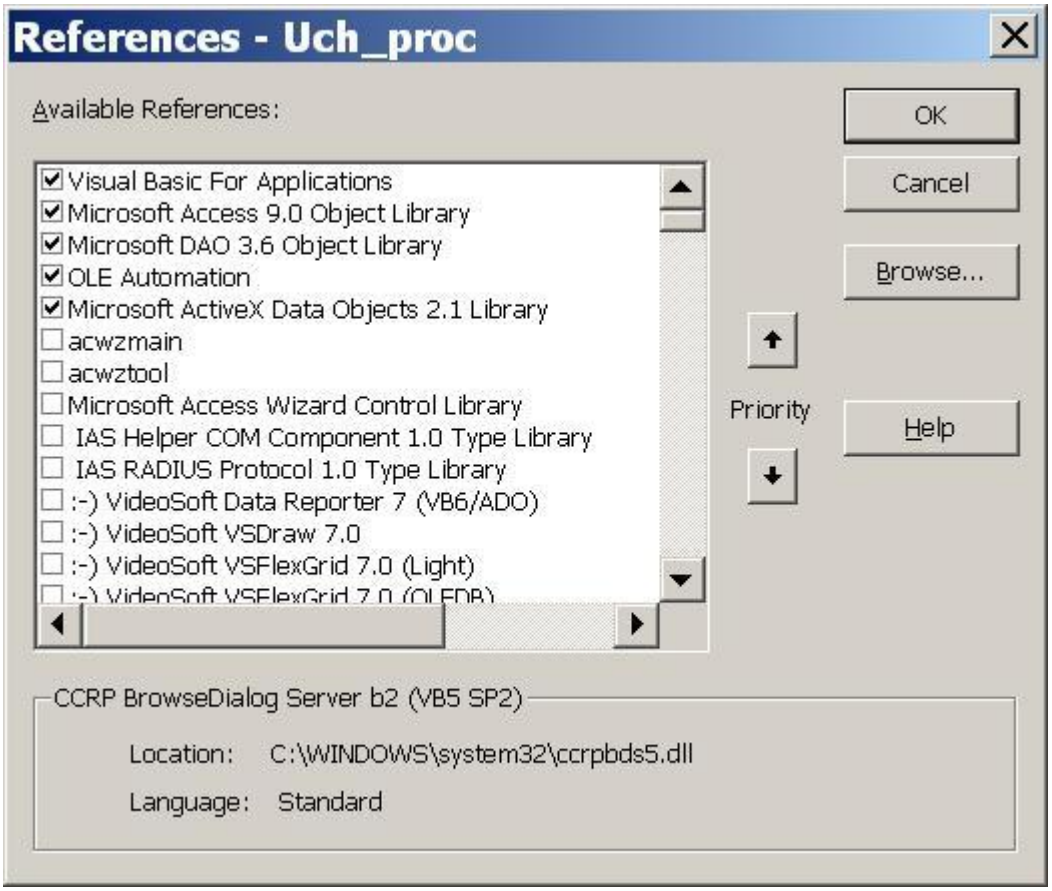


Рис. 6. Окно References

4.2 Листинг программы решения задачи

```
Option Compare Database 'Режим сравнения текстов
Option Explicit 'Режим явного описания переменных.

Private Sub Кнопка0_Click() 'Процедура обработки нажатия кнопки
  Dim dbs As Database  'Описание dbs как базы данных
  Dim rds As Recordset 'Описание rds как виртуальной таблицы
```

```
' Описание переменных вещественного типа
Dim A(100) As Single, MaxSum As Single, S As Single
' Описание переменных целого типа.
Dim i As integer, j As Integer, NumBegin As Integer
Dim NumEnd As Integer, N As Integer
Dim SL As String
Set dbs = CurrentDb 'Определить dbs как текущую базу данных
' Заполнить rds записями из таблицы "Последовательность"
SL = "SELECT * FROM Последовательность"
Set rds = dbs.OpenRecordset(SL)
rds.MoveLast 'Установить указатель на последнюю запись в rds
N = rds.RecordCount 'Определить количество записей в rds
rds.MoveFirst 'Установить указатель на первую запись в rds
For i = 1 To N 'Цикл, формирующий массив A
  A(i) = rds!Элемент 'Запомнили i-ое значение поля "Элемент".
  rds.MoveNext 'Перевели указатель на следующую запись в rds
Next 'Конец цикла
dbs.Close 'Закрыли базу данных
NumBegin = 1: NumEnd = 1    'Блок 3.1, рис. 3
MaxSum = A(1):S = 0: j = 1
For i = 1 To N              'Блок 4, рис. 2
  S = S + A(i)              'Блок 5.1, рис. 5
  If S > MaxSum Then        'Блок 5.2, рис. 5
    MaxSum = S: NumBegin = j: NumEnd = i 'Блок 5.3, рис. 5
  End If                    'Конец условия
  If S < 0 Then             'Блок 5.4, рис. 5
    j = i + 1: S = 0        'Блок 5.5, рис. 5
  End If                    'Конец условия
Next i                      'Конец цикла
Поле1.SetFocus              'Вывод решения
Поле1.Text = Str(MaxSum)    'путем установки
Поле3.SetFocus              'фокуса на элемент
Поле3.Text = Str(NumBegin)  'управления типа "поле"
Поле5.SetFocus              'и вывода в него
```

```
Поле5.Text = Str(NumEnd)    'строки с результатом.
End Sub                     'Конец процедуры
```

Код обработки события нажатия кнопки на форме состоит из трех основных частей:

- формирование массива A из записей таблицы ***Последовательность***;
- решение задачи;
- вывод результатов в текстовые поля.

Формирование массива выполнено через виртуальную таблицу rds, что является стандартным приемом работы с таблицами БД. Виртуальная таблица формируется при помощи инструкции ***SELECT***, являющейся командой языка **SQL (Structured Query Language)**, используемого для управления базами данных. VBA позволяет встраивать фрагменты SQL.

Для запуска программы следует открыть форму и щелкнуть по кнопке запуска. Результат выполнения программы показан на рис. 7.

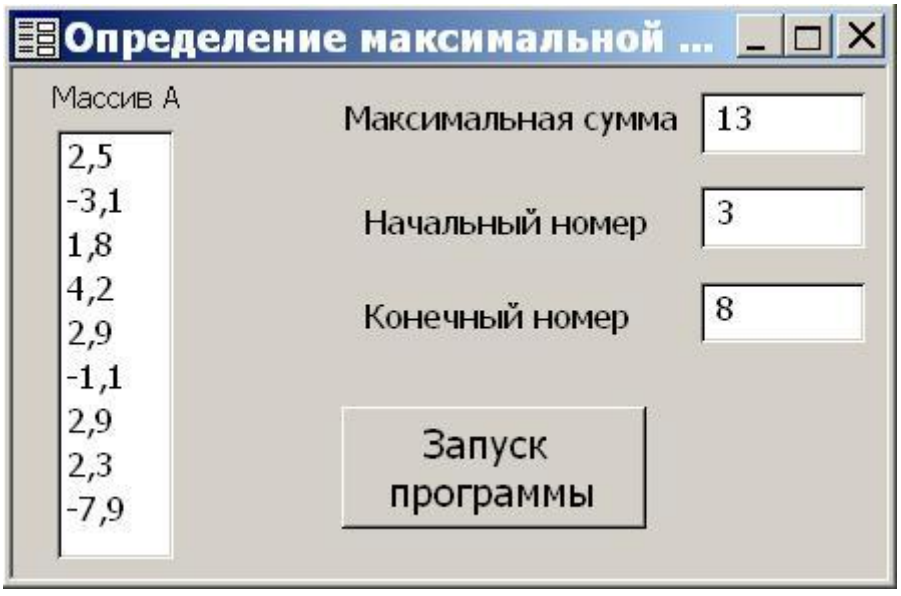


Рис. 7. Решение задачи

Контрольные вопросы

1. Какова главная идея концепции объектно-ориентированного программирования (ООП)?
2. Каковы основные характеристики объекта?

3. В чем отличие порядка выполнения программ в ООП и в процедурном программировании?
4. Каковы основные понятия ООП?
5. Что такое инкапсуляция? Приведите примеры.
6. Что означает наследование?
7. В чем заключается полиморфизм объектов?
8. Что такое класс? Что объединяет в себе класс?
9. Что такое объект? Приведите примеры объектов.
10. Как можно создать собственный класс в Visual Basic?
11. Из каких элементов состоит модуль класса?
12. Что такое элементы управления? Приведите примеры.
13. С чего начинается программирование в среде Access?
14. Как можно установить на форме элемент управления?
15. Как обеспечить выполнение элементом управления требуемых действий?
16. Каково назначение элементов управления надпись и поле?
17. Какова технология программирования с использованием элементов управления?
18. Что называется массивом? Как можно описать массив? Для чего служит описание массива?
19. Опишите этап формализации задачи на примере.
20. В чем смысл этапа моделирования?
21. Как выполняется разработка интерфейса приложения?
22. Как записать процедуру обработки события?
23. Из скольких частей состоит код обработки события нажатия кнопки на форме? Перечислите их.
24. Как запустить программу?

Оглавления.