ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИЙ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ
РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ
ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Курсовая работа

По предмету Программирование на C++

На тему: Свойства и виды алгоритмов

Студента 2в курса направления «Телекоммуникационные технологии»

Выполнил: Балтабаев Б.

Преподаватель: Ядгаров Ш.

Нукус 2014

Содержание

Введение

Применение компьютерных технологий в различных сферах современного общества станет значительно эффективнее, если пользователи овладеют системным подходом в решении прикладных задач, будут иметь представление о методах разработки алгоритмов и составления программ, а значит - о компьютеризации различных видов деятельности.

Процессор электронно-вычислительной машины, это чудо техники, умеет, тем не менее, выполнять лишь простейшие команды. Каким же образом компьютер решает сложнейшие задачи обработки информации? Для решения этих задач программист должен составить подробное описание последовательности действий, которые необходимо выполнить центральному процессору компьютера.

Составление такого пошагового описания процесса решения задачи называется алгоритмизацией, а алгоритмом называется конечный набор правил, расположенных в определённом логическом порядке, позволяющий исполнителю решать любую конкретную задачу из некоторого класса однотипных задач. В разных ситуациях в роли исполнителя может выступать электронное или какое-либо иное устройство или человек (например, военнослужащий, охраняющий склад боеприпасов и действующий согласно алгоритмам, записанным в устав караульной службы).

Глава 1. Алгоритм и его свойства. Способы записи алгоритма

Само слово «алгоритм» возникло из названия латинского перевода книги математика IX века Аль-Хорезми «Algoritmi de numero Indoru», что можно перевести как «Трактат Аль-Хорезми об арифметическом искусстве индусов». Составление алгоритмов и вопросы их существования являются предметом серьёзных математических исследований.

Под алгоритмом понимают набор правил, определяющих процесс преобразования исходных данных задачи в искомый результат.

Рассмотрим пример алгоритма для нахождения середины отрезка при помощи циркуля и линейки.

Алгоритм деления отрезка AB пополам:

1) поставить ножку циркуля в точку A;

2) установить раствор циркуля равным длине отрезка AB;

3) провести окружность;

4) поставить ножку циркуля в точку B;

5) провести окружность;

6) через точки пересечения окружностей провести прямую;

7) отметить точку пересечения этой прямой с отрезком AB.

Анализ примеров различных алгоритмов показывает, что запись алгоритма распадается на отдельные указания исполнителю выполнить некоторое законченное действие. Каждое такое указание называется командой. Команды алгоритма выполняются одна за другой. После каждого шага исполнения алгоритма точно известно, какая команда должна выполнятся следующей. Совокупность команд, которые могут быть выполнены исполнителем, называется системой команд исполнителя.

Алгоритм не только задает последовательность выполнения операций при решении конкретной задачи, но и должен обладать рядом свойств.

Свойства алгоритма:

- Однозначность алгоритма, под которой понимается единственность толкования исполнителем правила построения действий и порядок их выполнения. Чтобы алгоритм обладал этим свойством, он должен быть записан командами из системы команд исполнителя.
- Конечность алгоритма – обязательность завершения каждого из действий, составляющих алгоритм, и завершилось выполнения алгоритма в целом.
- Результативность алгоритма, предполагающая, что выполнение алгоритма должно завершиться получением определённых результатов.
- Массовость, т. е. возможность применения данного алгоритма для решения целого класса задач, отвечающих общей постановке задачи. Для того чтобы алгоритм обладал свойством массовости, следует составлять алгоритм, используя обозначения величин и избегая конкретных значений.
- Правильность алгоритма, под которой понимается способность алгоритма давать правильные результаты решения поставленных задач.
- Эффективность – для решения задачи должны использоваться ограниченные ресурсы компьютера (процессорное время, объём оперативной памяти и т. д.).

Создание алгоритма для решения задач какого-либо типа, его представление исполнителю в удобной для него форме – это творческий акт.

Алгоритм может быть представлен различными способами:

- на разговорном, естественном языке;
- на языке блок-схем;
- на языке программирования.

Выбор и разработка алгоритма и численного метода решения задачи имеют важнейшее значение для успешной работы над программой. Тщательно проработанный алгоритм решения задачи – необходимое условие эффективной работы по составлению алгоритму.

Приведем пример записи алгоритма на естественном языке, то есть на языке человеческого общения. Требуется вычислить сумму двух чисел.

Обозначим эти числа a и b. Тогда алгоритм можно записать следующим образом:

1. Считать число a.
2. Считать число b.
3. Выполнить суммирование c:= a + b.
4. Вывести число c.

Видно, что формулировка алгоритма не зависит от конкретных значений переменных a и b, поэтому его можно применять для решения достаточно большого числа сходных задач, в данном случае вместе составляющих целый класс задач суммирования. Алгоритм описывает действия не над конкретными значениями, а над абстрактными объектами.

Основными объектами программирования являются переменные. Переменные в программе отличаются от переменных, используемых в записи математических формул. Несмотря на сходство терминов, правила использования переменных в программах для компьютера отличаются от правил работы с математическими переменными. Это различие необходимо уяснить. В программировании переменную можно трактовать как одну или несколько ячеек оперативной памяти компьютера, которым присвоено определённое имя. Содержимое этих ячеек может меняться, но имя переменной остаётся неизменным. В математике значение переменной в рамках определённой задачи неизменно, но меняется в других задачах из данного класса. Именно поэтому конструкция a := a + 1 воспринимается программистом совершенно естественно, а уравнение a = a + 1 математик сочтёт неверным. В первом случае имеется в виду вычисление суммы содержимого ячейки a и числовой константы 1 и занесение полученного результата в ту же ячейку a. Второй случай равносилен неверному тождеству 0 = 1.

Иногда используют полуформальный язык с ограниченным словарём (часто на основе английского языка), промежуточный между естественным языком и языком программирования. Такой язык называется псевдокодом.

Запись алгоритма на псевдокоде называется структурным планом. Псевдокод удобен тем, что позволяет программисту сосредоточиться на формулировке алгоритма, не задумываясь над синтаксическими особенностями конкретного языка программирования.

Псевдокод:

Алгоритм < название >

Начало

< последовательность действий >

Конец

Любой алгоритм может быть представлен в виде последовательности действий. Под действием понимают либо базовую операцию, либо базовую структуру.

В качестве базовых операций используются:

- операция присваивания вида

< переменная > := < выражение >

- операция ввода/вывода

ввод (список ввода)

вывод (список вывода).

Смысл операции присваивания состоит в вычислении результата выражения, стоящего справа от знака «:=», для конкретных значений входящих в него переменных и присваивании этого результата переменной, стоящей слева от знака «:=», например:

D := 5

D := D+1

Min := C

При выполнении операции ввода ввод (A, B, C) переменным из списка ввода A, B и C присваиваются конкретные значения, вводимые с клавиатуры, например:

-5 7 20 {Enter}

В результате в памяти получим:

A = -5, B = 7, C = 20.

Операция вывода осуществляет вывод значений переменных и выражений из списка вывода на экран, например:

вывод (A, B, C, 10)

На экране получим:

- 5 7 20 10

Описание алгоритмов с помощью блок-схем.

Для разработки структуры программы удобнее пользоваться записью алгоритма в виде блок-схемы (в англоязычной литературе используется термин flow-chart). Для изображения основных алгоритмических структур и блоков на блок-схемах используют специальные графические символы.

Составим алгоритм вычисления квадратного корня из произвольного положительного вещественного числа x в виде блок-схемы.

Блок-схема для решения данного рода задач будет выглядеть следующим образом:

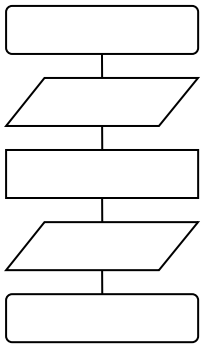
Начало

Ввод вещественного числа x

Вычисление корня по формуле

Вывод результата

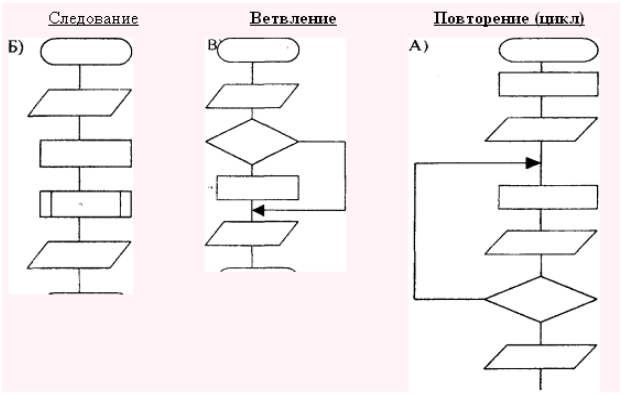
Конец



Глава 2. Классификация алгоритмов

Различают три типа базовых структур:

- Следование
- Развилка
- Цикл



Структура Следование - одна из самых важных структур. Она означает, что два действия должны быть выполнены друг за другом.

Структура Развилка обеспечивает выбор одной из двух альтернатив: если < условие 1 > то

< действие 1 >

иначе

< действие 2 >

все

Существует сокращенная форма структуры Развилка, которая позволяет выполнить действие или пропустить его:

если < условие > то < действие >

все

Обобщением структуры Развилка является Множественный выбор:

если (выражение 1) то < действие 1 >

если (выражение 2) то < действие 2 >

если (выражение 3) то < действие N >все

В зависимости от значения выражение выполняется одно из указанных действий, например, если (выражение 3), то выполняется < действие 3 >.

Третьей базовой структурой является Цикл, который предусматривает повторное выполнение определенных действий, необходимое для большинства программ. Различают следующие типы структур Цикл:

- цикл «от до»
- цикл «пока»
- цикл «до»

Цикл «от до» управляет повторением выполнения действия с помощью переменной цикла:

цикл от I:= N1 до N2

< действие >

кц

Здесь I - переменная цикла, N1, N2 - начальное и конечное значения переменной цикла, вычисляются один раз при входе в цикл. Переменная цикла пробегает все следующие друг за другом в порядке возрастания значения от начального до конечного. Изменение значения переменной цикла происходит автоматически после каждого выполнения действия, указанного внутри цикла. В зависимости от соотношения N1 и N2 цикл может не выполниться ни разу (N1>N2) или выполниться (N2-N1+1) раз.

В цикле «пока» управление внутри цикла осуществляется с помощью логического условия:

цикл пока < условие>

< действие >

кц

Выполнение действия повторяется до тех пор, пока истинно условие. Проверка условия осуществляется в начале цикла. Это означает, что действие может не выполниться ни разу. Чтобы такой цикл не был бесконечным, внутри цикла необходимо предусмотреть изменение значения условия с истинного на ложное.

Третий тип структуры цикл «до» имеет вид:

цикл

< действие > до < условие>

кц

Как только значение условия становится истинным, цикл прекращается. Цикл “до“ независимо от значения условия выполнится по меньшей мере один раз, т.к. проверка условия производится после выполнения действия. Для завершения цикла необходимо внутри цикла изменить условие с ложного на истинное. Выбор структуры цикла определяется особенностями алгоритма решения конкретной задачи.

Существенная особенность перечисленных базовых структур состоит в том, что каждая из них имеет один вход и один выход. Их можно соединять друг с другом в любой последовательности. В качестве действия может использоваться любая из перечисленных структур, что обеспечивает возможность вложенности одних структур в другие.

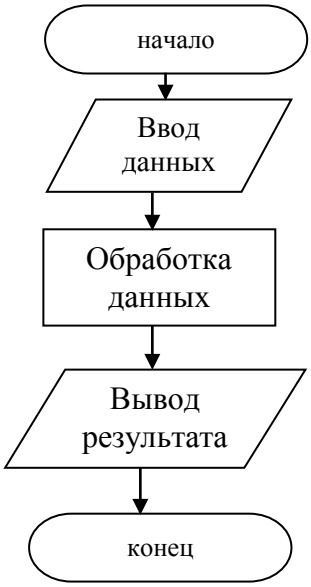
В зависимости от применяемых базовых структур различают следующие типы алгоритмов:

- линейные
- разветвляющиеся
- циклические.

2.1 Линейная алгоритмическая структура. Типовые примеры

Линейным называется алгоритм, блоки которого расположены последовательно один за другим, нет условий и повторений.

Покажем общую структуру линейного алгоритма в виде блок-схемы.



Основной принцип программирования заключается в том, что обрабатывать можно только те данные, которые находятся в определенных областях оперативной памяти компьютера. Для того чтобы поместить исходные данные в оперативную память используются операторы ввода данных.

Для реализации процесса обработки данных используется оператор присваивания.

Результат вычислений помещается в область S оперативной памяти. Чтобы вывести результат из памяти на экран монитора необходимо использовать оператор вывода.

Операторы ввода данных:

1. INPUT - оператор ввода данных с клавиатуры. Данные задаются в виде переменных. Переменная – это величина, значение которой может меняться в процессе выполнения программы. Для обозначения переменной используются их имена (идентификаторы) – последовательность до 40 латинских букв и цифр, начинающаяся с буквы. Данные могут быть следующих основных типов:

- целые INTEGER (Y%) – 2 байта в памяти (от -32768 до 32767),

- длинные целые LONG (Y&) – 4 байта (от -2^{31} до $2^{31}-1$),
- вещественные float – 6 знаков после , -4 байта (от -3.4E+38 до 3.4E+38),
- вещественные удвоенной точности DOUBLE -16 знаков после ,– 8 байт (от -E+308 до E+308),
- символьные STRING (Y$) – последовательность символов до 32767 символов длиной.

Например: INPUT a,b или INPUT “Введите два числа”;a,b

2. DATA, READ – операторы ввода данных из блока памяти. Например: DATA 3,4 : READ a,b

Оператор присваивания может быть использован как для ввода данных (Например: a=3 : b=4), так для вычисления выражений. (Например: S=a*b). Оператор присваивания вычисляет выражение, расположенное справа от символа присваивания (=) и результат присваивается переменной, расположенной слева от символа присваивания. При записи арифметического выражения используются арифметические операции и функции. Приоритет выполнения арифметических операций сохраняется. Функции можно использовать стандартные (встроенные) COS(X), sqrt(X) … и задаваемые самим пользователем. (Например: Y=3*sqrt(X)^2)

Для вывода данных используется оператор PRINT.

Например: cout<<S или cout “Площадь”;S или cout<< a,b,S

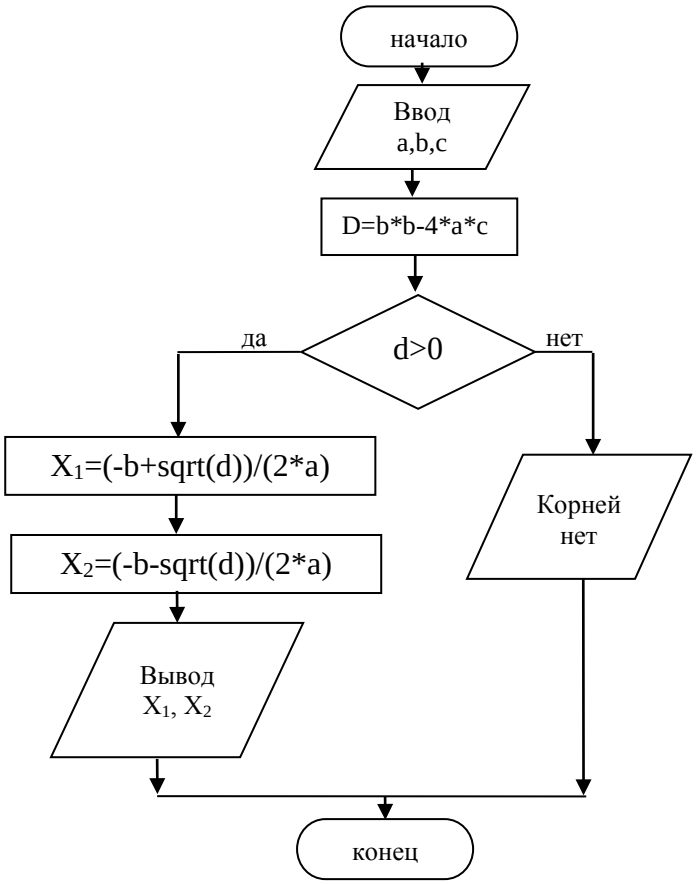
Пример линейной программы вычисления площади прямоугольника и ее алгоритм в виде блок-схемы:

```
cin>>“Введите две стороны прямоугольника”; a,b
S = a * b
cout<< “Площадь”; S
return 0;
```

2.2 Разветвляющая алгоритмическая структура. Основные операторы циклов. Типовые примеры

Алгоритм называется **разветвляющимся,** если содержит хотя бы одно условие, в результате которого обеспечивается переход на один из двух возможных вариантов решения задачи. Ветвление может быть полным (действия и после да и после нет) и неполным (в случае если нет – ничего не происходит).

Пример разветвляющегося алгоритма – алгоритм решения квадратного уравнения. Появление условия при решении этой задачи связано с отсутствием корней при отрицательном дискриминанте. Рассмотрим блок-схему этого алгоритма:



Для данной алгоритмической структуры характерно, что в любой момент времени её реализации осуществляется обработка только по какой-либо одной из ветвей.

Для описания разветвляющегося алгоритма существуют операторы:

условный

блочной структуры:

IF условие

блок действий 1

ELSE

блок действий 2

линейной структуры:

IF условие

блок 1

ELSE

блок 2

Обе структуры могут быть использованы как в полной форме так и в усеченной – без блока ELSE.

При работе условного оператора сначала проверяется выполнение условия. Если условие выполняется (истинное), то реализуется блок 1, в противном случае – блок 2. Условие – это логическое выражение, использующее операции сравнения (=, <, > <=, >=, <>) и логические операции (AND, OR).

Программа решения квадратного уравнения с использованием условного оператора имеет вид:

```
cin>> A,B,C ;
 D=B^2-4*A*C
IF (D>0)
X1=(-b+sqrt(d))/(2*a) : X2=(-b-sqrt(d))/(2*a) ;
cout<<X1” “X2
ELSE
```

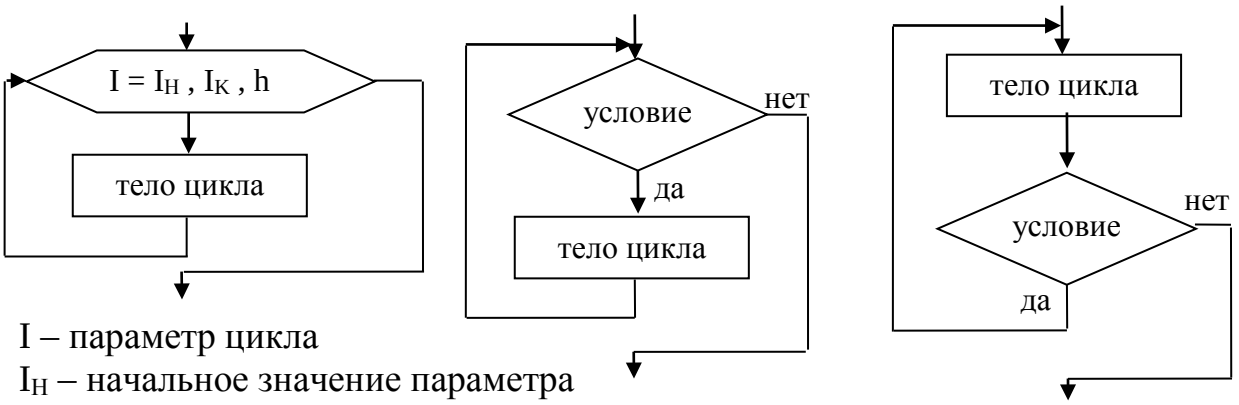
cout<<”Решением нет”

return 0;

2.3 Циклические алгоритмические структуры. Основные операторы ветвления. Типовые примеры

Алгоритм называется **циклическим,** если содержит участок, повторяющийся один или много раз.

Циклы бывают с определённым количеством, неопределённым числом вычислений.



I – параметр цикла
I_H – начальное значение параметра
I_K – конечное значение параметра
h – шаг изменения параметра (приращение)

Оператор цикла с параметром:

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

тело цикла

Оператор цикла с предусловием:

DO WHILE условие продолжения вычислений (UNTIL условие прекращения вычислений)

тело цикла

Оператор цикла с постусловием:

DO

тело цикла

Глава 3. Языки программирования

Прогресс компьютерных технологий определил процесс появления новых разнообразных знаковых систем для записи алгоритмов – языков программирования. Смысл появления такого языка – оснащенный набор вычислительных формул дополнительной информации, превращает данный набор в алгоритм.

Язык программирования служит двум связанным между собой целям: он дает программисту аппарат для задания действий, которые должны быть выполнены, и формирует концепции, которыми пользуется программист, размышляя о том, что делать. Первой цели идеально отвечает язык, который настолько "близок к машине", что всеми основными машинными аспектами можно легко и просто оперировать достаточно очевидным для программиста образом. Второй цели идеально отвечает язык, который настолько «близок к решаемой задаче», чтобы концепции ее решения можно было выражать прямо и коротко.

Связь между языком, на котором мы думаем/программируем, и задачами и решениями, которые мы можем представлять в своем воображении, очень близка. По этой причине ограничивать свойства языка только целями исключения ошибок программиста в лучшем случае опасно. Как и в случае с естественными языками, есть огромная польза быть, по крайней мере, двуязычным. Язык предоставляет программисту набор концептуальных инструментов, если они не отвечают задаче, то их просто игнорируют. Например, серьезные ограничения концепции указателя заставляют программиста применять вектора и целую арифметику, чтобы реализовать структуры, указатели и т.п. Хорошее проектирование и отсутствие ошибок не может гарантироваться чисто за счет языковых средств.

Может показаться удивительным, но конкретный компьютер способен работать с программами, написанными на его родном машинном языке.

Существует почти столько же разных машинных языков, сколько и компьютеров, но все они суть разновидности одной идей простые операции производятся со скоростью молнии на двоичных числах.

Персональные компьютеры IBM используют машинный язык микропроцессоров семейства 8086, т.к. их аппаратная часть основывается именно на данных микропроцессорах.

Можно писать программы непосредственно на машинном языке, хотя это и сложно. На заре компьютеризации(в начале 1950-х г.г.), машинный язык был единственным языком, большего человек к тому времени не придумал. Для спасения программистов от сурового машинного языка программирования, были созданы языки высокого уровня (т.е. немашинные языки), которые стали своеобразным связующим мостом между человеком и машинным языком компьютера. Языки высокого уровня работают через трансляционные программы, которые вводят "исходный код" (гибрид английских слов и математических выражений, который считывает машина), и в конечном итоге заставляет компьютер выполнять соответствующие команды, которые даются на машинном языке. Существует два основных вида трансляторов:

- интерпретаторы, которые сканируют и проверяют исходный код в один шаг,
- компиляторы, которые сканируют исходный код для производства текста программы на машинном языке, которая затем выполняется отдельно.

3.1 Эволюция и классификация языков программирования

Машинно – ориентированные языки - это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно –

ориентированные языки позволяют использовать все возможности и особенности Машинно – зависимых языков:

- высокое качество создаваемых программ (компактность и скорость выполнения);

- возможность использования конкретных аппаратных ресурсов;

- предсказуемость объектного кода и заказов памяти;

- для составления эффективных программ необходимо знать систему команд и особенности функционирования данной ЭВМ;

- трудоемкость процесса составления программ (особенно на машинных языках и ЯСК), плохо защищенного от появления ошибок;

- низкая скорость программирования;

- невозможность непосредственного использования программ, составленных на этих языках, на ЭВМ других типов.

Машинно-ориентированные языки по степени автоматического программирования подразделяются на классы.

Машинный язык. Отдельный компьютер имеет свой определенный Машинный язык (далее МЯ), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому МЯ является командным. Однако, некоторые семейства ЭВМ (например, ЕС ЭВМ, IBM/370/ и др.) имеют единый МЯ для ЭВМ разной мощности. В команде любого из них сообщается информация о местонахождении операндов и типе выполняемой операции.

В новых моделях ЭВМ намечается тенденция к повышению внутренних языков машинно – аппаратным путем реализовывать более сложные команды, приближающиеся по своим функциональным действиям к операторам алгоритмических языков программирования.

Языки Символического Кодирования. Продолжим рассказ о командных языках, Языки Символического Кодирования (далее ЯСК), так же, как и МЯ, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во

внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в ЯСК заменены на символы (идентификаторы), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ.

Использование символических адресов – первый шаг к созданию ЯСК. Команды ЭВМ вместо истинных (физических) адресов содержат символические адреса. По результатам составленной программы определяется требуемое количество ячеек для хранения исходных промежуточных и результирующих значений. Назначение адресов, выполняемое отдельно от составления программы в символических адресах, может проводиться менее квалифицированным программистом или специальной программой, что в значительной степени облегчает труд программиста.

Автокоды. Есть также языки, включающие в себя все возможности ЯСК, посредством расширенного введения макрокоманд - они называются Автокоды.

В различных программах встречаются некоторые достаточно часто использующиеся командные последовательности, которые соответствуют определенным процедурам преобразования информации. Эффективная реализация таких процедур обеспечивается оформлением их в виде специальных макрокоманд и включением последних в язык программирования , доступный программисту. Макрокоманды переводятся в машинные команды двумя путями – расстановкой и генерированием. В постановочной системе содержатся «остовы» - серии команд, реализующих требуемую функцию, обозначенную макрокомандой. Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в «остов» программы, превращая её в реальную машинную программу.

В системе с генерацией имеются специальные программы, анализирующие макрокоманду, которые определяют, какую функцию необходимо выполнить и формируют необходимую последовательность команд, реализующих данную функцию.

Обе указанных системы используют трансляторы с ЯСК и набор макрокоманд, которые также являются операторами автокода.

Развитые автокоды получили название Ассемблеры. Сервисные программы и пр., как правило, составлены на языках типа Ассемблер.

Макрос. Язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ на более сжатую форму - называется Макрос (средство замены).

В основном, Макрос предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется макропроцессором. На макропроцессор поступает макроопределяющий и исходный текст. Реакция макропроцессора на вызов-выдача выходного текста.

Макрос одинаково может работать, как с программами, так и с данными.

Машинно – независимые языки – это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и ВС.

Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка(задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на МЯ.

Т.о., командные последовательности (процедуры, подпрограммы), часто используемые в машинных программах, представлены в высокоуровневых языках отдельными операторами. Программист получил возможность не расписывать в деталях вычислительный процесс на уровне машинных команд, а сосредоточиться на основных особенностях алгоритма.

Проблемно – ориентированные языки. С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач, ими стали проблемно – ориентированные языки. Эти языки, языки ориентированные на решение определенных проблем, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме.

Проблемных языков очень много, например:

Фортран, Алгол – языки, созданные для решения математических задач;

Simula, Слэнг - для моделирования;

Лисп, Снобол – для работы со списочными структурами.

Универсальные языки. Универсальные языки были созданы для широкого круга задач: коммерческих, научных, моделирования и т.д. Первый универсальный язык был разработан фирмой IBM, ставший в последовательности языков Пл/1. Второй по мощности универсальный язык называется Алгол-68. Он позволяет работать с символами, разрядами, числами с фиксированной и плавающей запятой. Пл/1 имеет развитую систему операторов для управления форматами, для работы с полями переменной длины, с данными организованными в сложные структуры, и для эффективного использования каналов связи. Язык учитывает включенные во многие машины возможности прерывания и имеет соответствующие

операторы. Предусмотрена возможность параллельного выполнение участков программ.

Программы в Пл/1 компилируются с помощью автоматических процедур. Язык использует многие свойства Фортрана, Алгола, Кобола. Однако он допускает не только динамическое, но и управляемое и статистическое распределения памяти.

Диалоговые языки. Появление новых технических возможностей поставило задачу перед системными программистами – создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ их назвали диалоговыми языками.

Эти работы велись в двух направлениях. Создавались специальные управляющие языки для обеспечения оперативного воздействия на прохождение задач, которые составлялись на любых раннее неразработанных (не диалоговых) языках. Разрабатывались также языки, которые кроме целей управления обеспечивали бы описание алгоритмов решения задач.

Необходимость обеспечения оперативного взаимодействия с пользователем потребовала сохранения в памяти ЭВМ копии исходной программы даже после получения объектной программы в машинных кодах. При внесении изменений в программу с использованием диалогового языка система программирования с помощью специальных таблиц устанавливает взаимосвязь структур исходной и объектной программ. Это позволяет осуществить требуемые редакционные изменения в объектной программе.

Одним из примеров диалоговых языков является Бэйсик. Бэйсик использует обозначения подобные обычным математическим выражениям. Многие операторы являются упрощенными вариантами операторов языка Фортран. Поэтому этот язык позволяет решать достаточно широкий круг задач.

Непроцедурные языки. Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по

фиксированным алгоритмам (табличные языки и генераторы отчетов), и языков связи с операционными системами.

Позволяя четко описывать как задачу, так и необходимые для её решения действия, таблицы решений дают возможность в наглядной форме определить, какие условия должны быть выполнены прежде чем переходить к какому-либо действию. Одна таблица решений, описывающая некоторую ситуацию, содержит все возможные блок-схемы реализаций алгоритмов решения.

Табличные методы легко осваиваются специалистами любых профессий.

Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе.

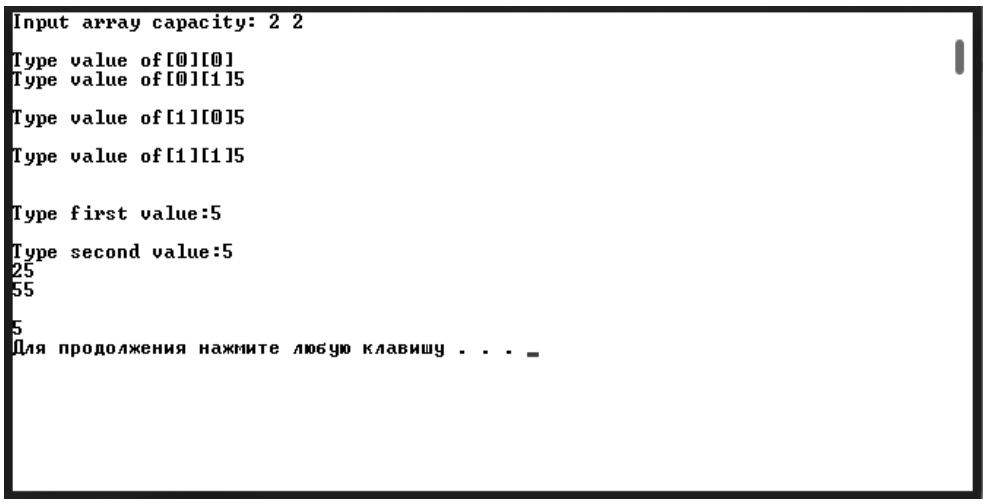
Практическая часть

1) Составьте программу которая создает матрицу?

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
   cout<<"Input array capacity:";
   int capacity;
   cin>>capacity;
   int **A=new int*[capacity];
   for(int i=0;i<capacity;i++)
         A[i]=new int[capacity];
   for(int i=0;i<capacity;i++)
         for(int j=0;j<capacity;j++)
         {
               cout<<"\nType value of["<<i<<"]["<<j<<"]";
               cin>>A[i][j];
               }
   cout<<endl;
   int lengthB=0;
   cout<<"\nType first value:";
   int K1;
   cin>>K1;
   cout<<"\nType second value:";
   int K2;
   cin>>K2;
   for(int i=0;i<capacity;i++)
   {
         for(int j=0;j<capacity;j++)
               {
```

```
                cout<<A[i][j];
                if(i>j&&A[i][j]>=K1&&A[i][j]<=K2)
                   ++lengthB;
             }
        cout<<endl;
   }
   int *B=new int[lengthB];
   int k=0;
   for(int i=1;i<capacity;i++)
        for(int j=0;j<capacity;j++)
        {
             if((i>j)&&A[i][j]>=K1&&A[i][j]<=K2)
             {
                                  B[k++]=A[i][j];
             }
             continue;
        }
        cout<<endl;
        for(int i;i<lengthB;i++)
             cout<<B[i]<<' ';
        cout<<endl;

   delete []B;
   for(int i=0;i<capacity;i++)
      delete [](A[i]);
   delete []A;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```



2) Даны три целых числа. Найти количество положительных и количество отрицательных чисел в исходном наборе.

```
#include  < iostream >
using namespace std ;
int main ( )
{
   int a , b , c ;
   int  x ; x = 0 ;
   int  y ; y = 0 ;
   cout << " a = " ; cin >> a ;
   cout << " b = " ;   cin >> b ;
   cout << " c = " ; cin >> c ;
   if ( a > 0 )
   x ++ ;
   if ( b > 0 )
   x ++ ;
   if ( c > 0 )
   x ++ ;
   cout  << "  x = " <<  x  <<  endl ;
   if ( a < 0 )
   y  ++ ;
   if ( b < 0 )
   y  ++ ;
   if ( c < 0 )
   y  ++ ;
   cout << " y = " << y << endl ;
   return 0 ;
}
```

```
a=1
b=1
c=-1
x=2
y=1

Process returned 0 (0x0)   execution time : 8.017 s
Press any key to continue.
```

3) Найти сумму $\sum_{n=1}^{10} \frac{1}{n^3}$

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{
  float S; S=0;
  for(int n=1; n<=10; ++n)
  S=S+1.0/pow(n,3);
  cout<<"S="<<S<<endl;
  return 0;
}
```

```
S=1.19753

Process returned 0 (0x0)   execution time : 0.234 s
Press any key to continue.
```

Заключение

Изобретение языка программирования высшего уровня позволило нам общаться с машиной, понимать её (если конечно Вам знаком используемый язык), как понимает американец немного знакомый с русским языком древнюю азбуку Кириллицы. Проще говоря, мы в нашем развитии науки программирования пока что с ЭВМ на ВЫ. Но если мы обратим внимание на темпы роста и развития новейших технологий в области программирования, то можно предположить, что в ближайшем будущем, человеческие познания в этой сфере, помогут произвести на свет языки, умеющие принимать, обрабатывать и передавать информации в виде мысли, слова, звука или жеста. Так и хочется назвать это детище компьютеризированного будущего: «языки программирования "высочайшего" уровня». Возможно, концепция решения этого вопроса проста, а ближайшее будущее этого проекта уже не за горами.

Размышляя над этим, хочется верить в прогресс науки и техники, в высоко - компьютеризированное будущее человечества, как единственного существа на планете, пусть и не использующего один, определенный разговорный язык, но способного так быстро прогрессировать и развивать свой интеллект, что и перехода от многоязыковой системы к всеобщему пониманию долго ждать не придется.

Единственный способ изучать новый язык программирования – писать на нём программы.

Список литературы

1. Коляда М. Г. Окно в удивительный мир информатики. – Д.: Сталкер, 1997.
2. “Языки программирования высокого уровня”, Хротко Г., 2002 г.;
3. “Языки программирования”, Малютин Э.А., Малютина Л.В., 2005 г.;
4. “Новые языки программирования и тенденции их развития”, Ушкова В., 2007 г.;
5. “Мир Лиспа” т.1, Хьювенен Э., Сеппенен Й., 1990 г.;
6. “Алгоритмические языки реального времени”, Янг С., 2005 г..
7. www.allbest.ru
8. www.cyberfirum.ru
9. www.informatika.ru