"ALGORITMLAR" FANIDAN O'QUV USLUBIY MAJMUA

O'ZBEKISTON RESPUBLIKASI OLIY VA O'RTA MAXSUS TA'LIM VAZIRLIGI

GULISTON DAVLAT UNIVERSITETI

"AXBOROT TEXOLOGIYALARI" kafedrasi

ALGORITMLAR FANIDAN

MARUZA

MASHG'ULOTLAR TO'PLAMI

Guliston- 2019 yil.

"ALGORITMLAR" FANIDAN O'QUV USLUBIY MAJMUA

Ushbu maruza mashg'ulotlari to'plami 5110700 - "Informatika o'qitish metodikasi" ta'lim yo'nalishi bakalavr talabalari uchun fan sifatida o'qitilayotgan "Algoritmlar" fan dasturi asosida yaratilgan. Undagi barcha mavzular, mazkur fan bo'yicha tavsiya etilgan adabiyotlar asosida olingan bo'lib, zamonaviy fan yutuqlarini va pedagogik tajribani hisobga olgan holda ishlab chiqilgan, hamda bo'lajak mutaxassis egallashi kerak bo'lgan bilim va ko'nikmalarni o'z ichiga oladi.

Tuzuvchilar: D.B.Abduraximov "Axborot texnologiyalari" kafedrasi mudiri, pedagogika fanlari nomzodi.

D.E.Abduraimov "Axborot texnologiyalari" kafedrasi o'qituvchisi

M.N.Normatova "Axborot texnologiyalari" kafedrasi o'qituvchisi

Ushbu maruza mashg'ulotlari to'plami "Axborot texnologiyalari" kafedrasining 20___-yil ___-_____dagi ___-sonli yig'ilishida muhokamadan o'tgan va ma'qullangan.

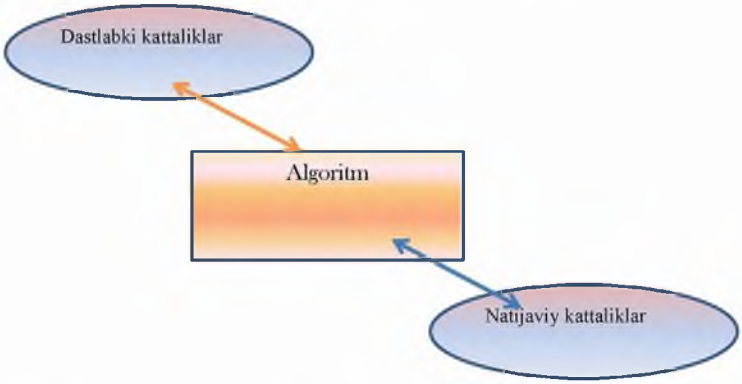
1-Modul. Algoritmlar tushunchasi va mohiyati.
MA'RUZA №1
MAVZU: ALGORITM TUSHUNCHASI VA ULARDAN FOYDALANISH.

REJA:

1. Algoritm tushunchasi. Algoritm xossalari.
2. Algoritm turlari va ularni tasvirlash usullari.
3. Algoritmik tillar.

Tayanch so'z va iboralar: algoritm, xossa, tasvirlash usullari, blok-sxema, dastur.

Algoritm bu aniq hisoblashlarni bajaruvchi protsedura bo'lib unga kirish qismida kattalik yoki kattaliklar berilib chiqishda natijaviy kattalik yoki kattaliklar olinadi. Demak algoritm hisoblovchi qadamlardan tashkil topgan bo'lib dastlabki qiymatlarga ko'ra natijaviy kattaliklar qiymatini beradi. Bu holatni sxematik tarzda quyidagicha tasvirlash mumkin.



Algoritmni qo'yilgan hisoblash masalani (computational problem) aniq bajaruvchi uskuna sifatida ham qaralishi mumkin. Algoritmlarda keltirilgan protseduralar yordamida kattaliklar bilan amallar bajarilib natijalar olinadi. Masalan, biror sonlar ketma-ketligini orta boorish tartibida saralash. Saralash masalasi (sorting problem) ga misol keltiramiz:

Kirish:n-ta sondan iborat sonlar ketma-ketligi ($a_1,a_2,\ldots,a_N$).
Chiqish: n-ta sondan iborat sonlar ketma-ketligi ($b_1 \le b_2 \ldots \le b_N$).
Misol, (31, 41, 59, 26, 41,56) kiruvchi ketma=ketlik bo'lsa, chiquvchi ketma-ketlik (26, 31, 41, 41, 56, 59) bo'lishi lozim. Bunga o'xshash kiruvchi ketma-ketlik saralash ekzemplyari (instanse) deb yuritiladi. Agar algoritm har qanday kiruvchi qiymatlar uchun aniq va mos chiquvchi qiymatlarni bera olsa

u aniq (correct) deb yuritiladi. Algorimlardan amaliyotda foydalanishga ayrim misollarni keltiramiz:

- Odam DNK si tarkibidagi 100 ming gen identifikatsiyasi, DNK-ni tashkil etuvchi 3 milliard asosiy juftlikni saralashva tahlili masalasi;
- Internetda ma'lumotlar olish masalasi: kata hajmdagi ma'lumotlarni olish, jo'natish,qidiruv va optimal marshrut tanlash;
- Electron kommertsiya masalalarida(kredit karta nomerlari , parollar, bank xisob-kitob raqamlari himoyasi, raqamli imzo va b);

Algoritmlarni ishlab chiqishda masalani yechimi uchun zarur bo'lgan vaqt va xotira hajmi muhim ko'rsatgichlar hisoblanib algoritmlarni yaratishda ularni samarali foydalanishni hisobga olish zarur. Aynan bir masalani yechish uchun turli algoritmlar tuzilishi mumkin. Ular bir-biridan samardorlik darajasi bilan farqlanadilar. Bu farq turli texnik va dasturiy ta'minotlarda har xil bo'lishi mumkin.

Misol uchun ikkita saralash algorutmlari farqini ko'rib chiqamiz:

Saralash algoritmi	Sarflanadigan vaqt	Izoh
Joylashtirish usuli	C_1n^2 bu N^2-ga propor-tsional	C_1-n ga bog'liq bo'lmagan doimiylik n-saralanadigan elementlar soni
Qo'shish usuli	C_2nlgn	lgn=$\log_2$n, C_2-n ga bog'liq bo'lmagan doimiylik

Qo'shish usuli joylashtirish usulidan samaraliroq ekanligini quyida keltirilgan jadval ma'lumotlarini tahlili orqali keltiramiz.

komputerlar	Saralanadigan sonlar soni	Saralovchi algoritm	Talab qilinadigan vaqt
A(tez ishlovchi 1sekundda 10mlrd amal bajaradi)	10 mlnta(taqribaan80 mb)	Joylashtirish usuli (tajribali dasturchi tomonidan yaratilgan algoritm saralash uchun $2n^2$ amal bajariladi)	$\frac{2*(10^7)^2 \ buyruqlar}{10^{10}\ buyruq/sec}$ $=20000\ sec$ *(5,5 soatdan ko'proq)*
B(sekin ishlovch-1sekundda		Qo'shish usuli (o'rta	$\frac{50*10^7\ lg10^7}{10^7} \approx 1163\ sekund$

10 mln amal bajaradi)		darajali dasturchi tomonidan yaratilgan algoritm saralash uchun 50nlgnamal bajariladi))	(20 min *dan kam*)

Umuman olganda algoritm - bu quyilgan masalaning echimiga olib keladigan, ma'lum qoidaga binoan bajariladigan amallarning chekli qadamlar ketma-ketligidir. Boshqacha qilib aytganda algoritm boshlang'ish ma'lumotlardan natijagasha olib keluvshi jarayonning aniq yozilishidir.

Algoritm tushunshasining turli ta'riflari bir qator talablarga javob berishi kerak:

- algoritm chekli sondagi elementar bajariluvshi ko'rsatmalardan iborat bo'lishi kerak;
- algoritm chekli sondagi qadamlardan iborat bo'lishi kerak;
- algoritm barsha boshlang'ish berilganlar ushun umumiy bo'lishi kerak;
- algoritm to'g'ri echimga olib kelishi kerak.

Har qanday algoritm ma'lum ko'rsatmalarga binoan bajariladi va bu ko'rsatmalarga buyruq deyiladi. Yuqoridagi fikrga ko'ra algoritm asosan masalani eshimini toppish ushun tuziladi.

Bitta masalani eshishning bir neshaalgoritmi mavjud bo'lishi mumkin. Ular orasida eng samaralisini, bajarilishi ushun eng kam amallar, mashina vaqti, xotira va h.k.ni talab qiluvshi algoritmni tanlash lozim. Samarali algoritmlar mavjud bo'lishi shartlari va ularni qurish (ishlab shiqish)ni o'rganish algoritmlar nazariyasi asosini tashkil etadi.

Algoritm kibernetika va matematikaning asosiy tushunshalaridan biri bo'lib bu atama o'rtaasrlarda yashab ijod etgan buyuk o'zbek matematigi Al-Xorazmiy nomidan kelib shiqqan. U IX asrning 825 yilidayoq o'zi kashf etgan o'nli sanoq tizimida to'rt arifmetikaamallarini bajarish qoidalarini bergan. Arifmetikaamallarini bajarish jarayoni esaalxorazm deb atalgan. Bu atama 1747 yildan boshlab algorismus, 1950 yilga kelib algorifm deb ham ataldi. Fanda "Yevklid algoritmi", "G'iyosiddin Koshiy algoritmi", "Laure algoritmi", "Markov algoritmi" deb ataluvshi algoritmlar ma'lum algoritm tushunshasi tobora kengayib borib, kibernetikaning nazariy va mantiqiy asosi hisoblangan algoritmlar nazariyasi paydo bo'lgan. Kompyuterlar paydo bo'lishi bilan algoritm atamasi hozirgi ma'nosi bilan axborot texnologiyalari sohasida eng asosiy atamalardan biri bo'lib qoldi. Odatda algoritmlar u yoki bu hisoblashga doir masalalarni (somputational problems) eshish ushun tuziladi.

Qo'yilgan masala ushun yaratiladigan algoritmda kiruvshi va shiquvshi ma'lumotlar muhim ahamiyatga ega, agar algoritm to'g'ri tuzilgan bo'lsa, ijroshi (kompyuter) aniq natijalar beradi.

Algoritm quyidagi xossalarga ega: aniqlik, tushunarlilik, ommaviylik, natijaviylik va diskretlik.

Aniqlik va tushunarlilik - degandaalgoritmda ijroshiga berilayotgan ko'rsatmalar aniq mazmunda bo'lishi tushuniladi. SHunki ko'rsatmalardagi noaniqliklar mo'ljallangan maqsadga erishishgaolib kelmaydi. Ijroshiga tavsiya etiladigan ko'rsatmalar tushunarli mazmunda bo'lishi shart, aks holda ijroshi uni bajaraolmaydi.

Ommaviylik - deganda har bir algoritm mazmuniga ko'ra bir turdagi masalalarning barshasi ushun ham o'rinli bo'lishi, ya'ni umumiy bo'lishi tushuniladi.

Natijaviylik –deganda algoritmda chekli qadamlardan so'ng albatta natija bo'lishi tushuniladi. Shuni ta'kidlash joizki, algoritm avvalfdan ko'zlangan maqsadga erishishga olib kelmasligi ham mumkin. Bunga ba'zan algoritmning noto'g'ri tuzilgani yoki boshqa xatolik sabab bo'lishi mumkin, ikkinshi tomondan, qo'yilgan masala ijodiy yeshimga ega bo'lmasligi ham mumkin. Lekin salbiy natija ham deb qabul qilinadi.

Diskretlik – deganda algoritmlarni chekli qadamlardan tashkil qilib bo'laklash imkoniyati tushuniladi.

Algoritmlarga doir quyidagi masalalarni misol sifatida keltirish mumkin:

- Talabani kundalik ishlarni tashkil etish;
- To'rtburshak piremetri va yuzasini hisoblash;
- R radusli doirani yuzasini va aylana uzunligini toppish;
- $A_1, A_2, A_3, \ldots A_n$ sonlarni toq elementlarini yig'indisini toppish;
- Berilgan ketma-ketlik sonlarni o'sish (kamayish) tartibda joylashtirish va h.k.

Algoritmning ushta turi mavjud: shiziqli, tarmoqlanuvshi va takrorlanuvshi(tsiklik).

CHiziqli algoritmlar - hesh qanday shartsiz faqat ketma-ket bajariladigan jarayonlardir.

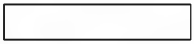
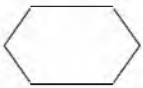
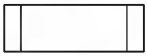
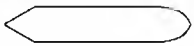
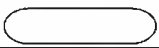
Tarmoqlanuvshi algoritmlar - ma'lum shartlarga muvofiq bajariladigan jarayonlardir.

Takrorlanuvshi algoritmlar - biron bir shart tekshirilishi yoki biron parametrning har xil qiymatlari asasidachekli ravishda takrorlanish yuz beradigan jarayonlardir.

Algoritmlarni turli usullarda tasvirlash mumkin.

- so'z bilan ifodalash;
- formulalarda berish;
- blok-sxemalarda tasvirlash;
- dastur shaklida ifodalash va boshqalar.

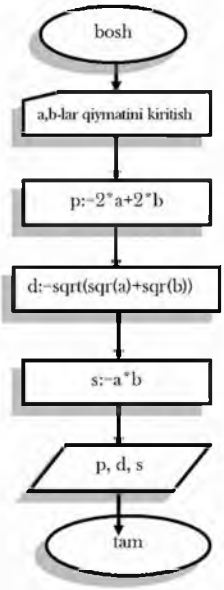
Algoritmlarni blok-sxema ko'rinishda tasvirlash qulay va tushunarli bo'lgani ushun eng ko'p ishlatiladi. Bunda algoritmdagi har bir ko'rsatma o'z shakliga ega. Masalan: parallelogramm ko'rinishdagi belgi ma'lumotlarni kiritish va shiqarish; to'g'ri to'rtburshak belgisi hisoblash jarayonini; romb belgisi shartlarning tekshirilishini bildiradi. Algoritimni blok-sxema shaklida tasvirlashda quyidagi geometrik figuralardan foydalaniladi:

Nomi	Belgilanishi	Bajaradigan vazifasi
Jarayon		Bir yoki bir nechta amallarni bajarilishi natijasida ma'lumotlarning uzgarishi
Karor		Biror shartga boglik ravishda algoritmning bajarilish yunalishini tanlash
SHakl uzgartirish		Dasturni uzgartiruvchi buyruk yoki buyruklar turkumini uzgartirish amalini bajarish
Avval aniklangan jarayon		Oldindan ishlab chikilgan dastur yoki algoritmdan foydalanish
Kiritish CHikarish		Axborotlarni kayta ishlash mumkin bulgan shaklga utkazish yoki olingan natijani tasvirlash
Displey		EXMga ulangan displeydan axborotlarni kiritish yoki chikarish
Xujjat		Axborotlarni kogozga chikarish yoki kogozdan kiritish
Axborotlar okimi chizigi		Bloklar orasidagi boglanishlarni tasvirlash
Boglagich		Uzilib kolgan axborot okimlarini ulash belgisi
Boshlash Tugatish		Axborotni kayta ishlashni boshlash, vaktincha yoki butunlay tuxtatish
Izoh		Bloklarga tegishli turli xildagi tushuntirishlar

Algoritmlarni tasvirlash usullariga misollar keltirib o'tamiz:

Masala: to'g'ri to'rtburshakning tomonlariga ko'ra uning perimetri, diagonali va yuzasini hisoblash.

1. So'z bilan ifodalash:
   - 1.1. boshlash;
   - 1.2. tomonlar qiymatini kiritish (a, b);
   - 1.3. perimetr qiymatini hisoblash (p);
   - 1.4. diagonal qiymatini hisoblash (d);
   - 1.5. yuzasini hisoblash (s);
   - 1.6. perimeter, diagonal va yuzasini qiymatini shop etish.
2. Formulalarda berish:
   - 2.1. A va B to'rtburshak tomonlari qiymatlari;
   - 2.2. *P=2*a+2*b;*
   - 2.3. $D=\sqrt{a^2+b^2}$;
   - 2.4. S=a*b;
   - 2.5. P, D va S qiymatlarini shop etish
3. Blok-sxemalarda tasvirlash:



4. Dastur shaklida ifodalash: (Pascal dasturlash tili misolida)

Program to'rtburshak yuzi;
Var a, b: Integer;
P, d, s: real;
Begin
Write ('a,btomonlarni qiymatlari kiritilsin');

```
ReadLn(a,b);
P:=2*a+2*b;
D:=sqrt(sqr(a)+sqr(b));
S:=a*b;
WriteLn('to'rtburshak perimetri=',p);
WriteLn('to'rtburshak dioganperli=',d);
WriteLn('to'rtburshak yuzasi=',S);
End.
```

Hozirgi kunda juda ko'p algoritmik tillar mavjud bo'lib, ularni dasturlash tillari deb ataymiz. Algoritmik til - algoritmlarni bir xil vaaniq yozish ushun ishlatiladigan belgilashlar va qoidalar tizimidir. Algoritmik til oddiy tilga yaqin bo'lib u matematik belgilarni (yuqorida aytilganidek) o'z ishigaoladi. Qo'yilgan masalalarni eshishga tuzilgan algoritmlarni to'g'ridan-to'g'ri mashinaga berib, eshib bo'lmaydi, shu sababli yozilgan algoritmni biror bir algoritmik tilga o'tkazish zarur. Har qanday algoritmik til o'z qo'llanilish sohasiga ega. Masalan, o'quv jarayonlari ushun Pascal, Delphi, VBA, java, C++dasturlash tillari va boshqalar.

1-misol: Kiritilgan n-natural sonni tub ko'paytuvchilarga ajratuvchi algoritmni Pascal dasturlash tilida ifodalanishini ko'rib chiqamiz:

```
var i,k:integer;   n:integer;
a:array[byte] of integer; label qq;
procedure opr(nn:integer);
begin  i:=2;
while(nn>0) do
begin
if nn  mod i=0 then  write(i,' ');
i:=i+1;
nn:=nn div i;
end;
end;
begin
readln(n);
k:=1;
for i:=2 to n do
while  (n  mod i=0) do
begin    a[k]:=i;  write(a[k],' ');
n:=n div i ;  k:=k+1;  end;
writeln;
readln;
end.
```

2-misol: $\int_0^{\pi} x^2 dx$ – qiymatini hisoblovchi dastur tuzing.

```
procedure
TForm1.Button1Click(Sender:
TObject);
var
h,a,x,d,b,s:real;
n:integer;
begin
a:=0;   s:=0;
b:=5;
n:=10000;
h:=(b-a)/n;
x:=a;
while (x<b) do
begin
d:=sqr(x);
s:=s+d*h;
x:=x+h;
end;
end;
end.
```

Chapter 24), and using a search engine to quickly find pages on which particular information resides (related techniques are in Chapters 11 and 32).

- Electronic commerce enables goods and services to be negotiated and exchanged electronically, and it depends on the privacy of personal information such as credit card numbers, passwords, and bank statements. The core technologies used in electronic commerce include public-key cryptography and digital signatures (covered in Chapter 31), which are based on numerical algorithms and number theory.
- Manufacturing and other commercial enterprises often need to allocate scarce resources in the most beneficial way. An oil company may wish to know where to place its wells in order to maximize its expected profit. A political candidate may want to determine where to spend money buying campaign advertising in order to maximize the chances of winning an election. An airline may wish to assign crews to flights in the least expensive way possible, making sure that each flight is covered and that government regulations regarding crew scheduling are met. An Internet service provider may wish to determine where to place additional resources in order to serve its customers more effectively. All of these are examples of problems that can be solved using linear programming, which we shall study in Chapter 29.

Although some of the details of these examples are beyond the scope of this book, we do give underlying techniques that apply to these problems and problem areas. We also show how to solve many specific problems, including the following:

- We are given a road map on which the distance between each pair of adjacent intersections is marked, and we wish to determine the shortest route from one intersection to another. The number of possible routes can be huge, even if we disallow routes that cross over themselves. How do we choose which of all possible routes is the shortest? Here, we model the road map (which is itself a model of the actual roads) as a graph (which we will meet in Part VI and Appendix B), and we wish to find the shortest path from one vertex to another in the graph. We shall see how to solve this problem efficiently in Chapter 24.
- We are given two ordered sequences of symbols, $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, and we wish to find a longest common subsequence of X and Y. A subsequence of X is just X with some (or possibly all or none) of its elements removed. For example, one subsequence of $\langle A, B, C, D, E, F, G \rangle$ would be $\langle B, C, E, G \rangle$. The length of a longest common subsequence of X and Y gives one measure of how similar these two sequences are. For example, if the two sequences are base pairs in DNA strands, then we might consider them similar if they have a long common subsequence. If X has m symbols and Y has n symbols, then X and Y have 2^m and 2^n possible subsequences.

respectively. Selecting all possible subsequences of X and Y and matching them up could take a prohibitively long time unless m and n are very small. We shall see in Chapter 15 how to use a general technique known as dynamic programming to solve this problem much more efficiently.

- We are given a mechanical design in terms of a library of parts, where each part may include instances of other parts, and we need to list the parts in order so that each part appears before any part that uses it. If the design comprises n parts, then there are $n!$ possible orders, where $n!$ denotes the factorial function. Because the factorial function grows faster than even an exponential function, we cannot feasibly generate each possible order and then verify that, within that order, each part appears before the parts using it (unless we have only a few parts). This problem is an instance of topological sorting, and we shall see in Chapter 22 how to solve this problem efficiently.
- We are given n points in the plane, and we wish to find the convex hull of these points. The convex hull is the smallest convex polygon containing the points. Intuitively, we can think of each point as being represented by a nail sticking out from a board. The convex hull would be represented by a tight rubber band that surrounds all the nails. Each nail around which the rubber band makes a turn is a vertex of the convex hull. (See Figure 33.6 on page 1029 for an example.) Any of the 2^n subsets of the points might be the vertices of the convex hull. Knowing which points are vertices of the convex hull is not quite enough, either, since we also need to know the order in which they appear. There are many choices, therefore, for the vertices of the convex hull. Chapter 33 gives two good methods for finding the convex hull.

These lists are far from exhaustive (as you again have probably surmised from this book's heft), but exhibit two characteristics that are common to many interesting algorithmic problems:

1. They have many candidate solutions, the overwhelming majority of which do not solve the problem at hand. Finding one that does, or one that is "best," can present quite a challenge.
2. They have practical applications. Of the problems in the above list, finding the shortest path provides the easiest examples. A transportation firm, such as a trucking or railroad company, has a financial interest in finding shortest paths through a road or rail network because taking shorter paths results in lower labor and fuel costs. Or a routing node on the Internet may need to find the shortest path through the network in order to route a message quickly. Or a person wishing to drive from New York to Boston may want to find driving directions from an appropriate Web site, or she may use her GPS while driving.

Not every problem solved by algorithms has an easily identified set of candidate solutions. For example, suppose we are given a set of numerical values representing samples of a signal, and we want to compute the discrete Fourier transform of these samples. The discrete Fourier transform converts the time domain to the frequency domain, producing a set of numerical coefficients, so that we can determine the strength of various frequencies in the sampled signal. In addition to lying at the heart of signal processing, discrete Fourier transforms have applications in data compression and multiplying large polynomials and integers. Chapter 30 gives an efficient algorithm, the fast Fourier transform (commonly called the FFT), for this problem, and the chapter also sketches out the design of a hardware circuit to compute the FFT.

Data structures

This book also contains several data structures. A *data structure* is a way to store and organize data in order to facilitate access and modifications. No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them.

Technique

Although you can use this book as a "cookbook" for algorithms, you may someday encounter a problem for which you cannot readily find a published algorithm (many of the exercises and problems in this book, for example). This book will teach you techniques of algorithm design and analysis so that you can develop algorithms on your own, show that they give the correct answer, and understand their efficiency. Different chapters address different aspects of algorithmic problem solving. Some chapters address specific problems, such as finding medians and order statistics in Chapter 9, computing minimum spanning trees in Chapter 23, and determining a maximum flow in a network in Chapter 26. Other chapters address techniques, such as divide-and-conquer in Chapter 4, dynamic programming in Chapter 15, and amortized analysis in Chapter 17.

Hard problems

Most of this book is about efficient algorithms. Our usual measure of efficiency is speed, i.e., how long an algorithm takes to produce its result. There are some problems, however, for which no efficient solution is known. Chapter 34 studies an interesting subset of these problems, which are known as NP-complete.

Why are NP-complete problems interesting? First, although no efficient algorithm for an NP-complete problem has ever been found, nobody has ever proven

that an efficient algorithm for one cannot exist. In other words, no one knows whether or not efficient algorithms exist for NP-complete problems. Second, the set of NP-complete problems has the remarkable property that if an efficient algorithm exists for any one of them, then efficient algorithms exist for all of them. This relationship among the NP-complete problems makes the lack of efficient solutions all the more tantalizing. Third, several NP-complete problems are similar, but not identical, to problems for which we do know of efficient algorithms. Computer scientists are intrigued by how a small change to the problem statement can cause a big change to the efficiency of the best known algorithm.

You should know about NP-complete problems because some of them arise surprisingly often in real applications. If you are called upon to produce an efficient algorithm for an NP-complete problem, you are likely to spend a lot of time in a fruitless search. If you can show that the problem is NP-complete, you can instead spend your time developing an efficient algorithm that gives a good, but not the best possible, solution.

As a concrete example, consider a delivery company with a central depot. Each day, it loads up each delivery truck at the depot and sends it around to deliver goods to several addresses. At the end of the day, each truck must end up back at the depot so that it is ready to be loaded for the next day. To reduce costs, the company wants to select an order of delivery stops that yields the lowest overall distance traveled by each truck. This problem is the well-known "traveling-salesman problem," and it is NP-complete. It has no known efficient algorithm. Under certain assumptions, however, we know of efficient algorithms that give an overall distance which is not too far above the smallest possible. Chapter 35 discusses such "approximation algorithms."

Parallelism

For many years, we could count on processor clock speeds increasing at a steady rate. Physical limitations present a fundamental roadblock to ever-increasing clock speeds, however: because power density increases superlinearly with clock speed, chips run the risk of melting once their clock speeds become high enough. In order to perform more computations per second, therefore, chips are being designed to contain not just one but several processing "cores." We can liken these multicore computers to several sequential computers on a single chip; in other words, they are a type of "parallel computer." In order to elicit the best performance from multicore computers, we need to design algorithms with parallelism in mind. Chapter 27

[1]

FOYDALANILGAN ADABIYOTLAR RO‘YHATI:

1. Thomas H. Cormen va b. Intruduction to algorithms. Massachusetts Institute of Technology. London 2009.(5-10pp)
2. Thomas H. Cormen va b. Intruduction to algorithms. Massachusetts Institute of Technology. London 2009. (11-13pp)
3. Слинкин Д.А.Основы программирования на Турбо-Паскале: Учебно-методическое пособие для студентов вузов. Шадринск: Изд-во Шадрииского пединститута, 2003. - 244 с. (10 -p)
4. M.U.Ashurov, N.D.Mirzaxmedova .Turbo Pascal dasturlash tili.(Uslubiy qo‘llanma),Toshkent TDPU – 2011 (3-10pp)

[1] Thomas H. Cormen va b. Intruduction to algorithms. Massachusetts Institute of Technology. London 2009.(5-10pp)

MA'RUZA №2

MAVZU: ALGORITMLAR SAMARADORLIGINI BAHOLASH.

Reja:

1. Algoritmlar samaradorligini baholash.
2. Algoritmlar va boshqa texnologiyalar.
3. Tarixiy ma'lumotlar.
4. Algoritmlar nazariyasi fanining predmeti, maqsadi va vazifalari.
5. Algoritmlar nazariyasining nazariy va amaliy ahamiyati.

Kalit so'zlar: Alan Tyuring, Aloiz Chyorch, Emil Post, Tyuring mashinasi, Post mashinasi, Chyorchning lyamda- hisoblanuvchanlik usuli, P=NP muammosi.

1. Algoritmlar samaradorligini baholash.

Kompyuteringizning tezligi va xotira miqdorini abadiy oshirish mumkin, deylik. Bu holatda algoritmlar o'rganish kerakmi? Bor bo'lishi mumkin, lekin faqat namoyish etish uchun, echim usulini cheklangan vaqti bor va u to'g'ri javob beradi.

Kompyuterlar juda tez bo'lganda, masalani echishga har qanday konkret usul mos kelarmidi.

Albatta, bugungi kunda juda samarali kompyuterlar, lekin ularning ishlashi juda katta bo'lishi mumkin emas. Xotira ham arzon, lekin bepul bo'lishi mumkin emas. Shunday qilib, hisob-vaqti - cheklangan resurs, shuningdek xotira miqdori ham. Siz donolik bilan bu resurslarini boshqarishingiz kerak,bunga algoritmlardan, vaqt va xotira xarajatlaridan samarali foydalanish kerak.

Har xil masalalarni yechish uchun mo'ljallangan, turli xil algoritmlar, samaradorligi bo'yicha sezilarli darajada farq qiladi. Bu farqlar juda katta bo'lishi mumkin ekan. Masalan, ikki saralash algoritmlar, 2-darsta muhokama qilinadi. Birinchisini bajarish uchun, saralashni joylashtirish, bunga vaqt kerk bo'ladi, shunday baholanmoqda c_1n^2, n- bu saralash elementlarning soni, c_1 bo'lsa bu – doimiy, n ga bog'liq emas. Shunday qilib, bu algoritmni vaqti taxminan n2 proportsional.

Ikkinchi algoritm amalga oshirish uchun, saralash birlashtirishi, vaqt talab etadi, taxminan c_2n lg n ga teng, lg n- bu $\log_2 n$ qisqa yozuvi, c_2 bu - boshqa doimiy n ga bog'liq emas.Odatda doimiy usul qo'shimchalar doimiy birlashish usulidan kichikroq, $c_1 < c_2$. Doimiy omillar algoritmni ish vaqtiga juda kan ta'sir qiladigan bog'liq omillardan ko'ra, shunga ishonch hosil qilaylik.Saralashni joylashtirish algoritmni ish vaqtini shunday yozaylik c_1 n n,birlashtirish saralashini esa c_2n lgn.

Joylashtirish saralashi n omilga ega, birlashtirish saralashi esa lg n ga ega bu esa sezarli darajada kamligini ko'rishimiz mumkin.Kiritish hajmi n etarlicha katta bo'lganda qo'shish saralashi odatda tezroq bo'ladi, saralash ob'ektlar kichik hajmdagi birlashtirishda, katta n uchun ahamiyatsiz qiymati lgn nisbatan n to'liq doimiy farqi qadriyatlar o'rnini qoplash, aslida birlashish yanada sezilarli namoyon bo'ladi, saralash afzalligi ziyoda.Bu doimiy c_1, c_2

dan necha marta kam muhim emas.Saralash elementlarini sono ishshi bilan burilish nuqtasi hosil bo'ladi,shunda birlashish saralashi yanada samarali bo'ladi.

Misol tarzida ikkita A va B kompyuterlarni ko'rib chiqamiz.A kompyuteri ancha tezroq, va unda joylashtirish saralashi algoritmi ishlaydi, B kompyuter esa sekin va unda saralash algoritmi birlashtirish usuli bilan ishlaydi.Har ikkita kompyuterlar bir nechta saralashni bajarishi kerak.Kompyuter A sekundiga o'n milliard ko'rsatmalar bajaradi, B kompyuter sekundiga faqat o'n million ko'rsatmalar bajaradi,shuhday qilib A kompyuteri ming marta B kompyuterdan tez. Saralash birlashishi yuqori darajadagi til yordamida bir programct tomonidan amalga oshirilgan.

Bu kompilyator juda samarali emas edi, va natija 50n lg n ko'rsatmalarga bajaradigan kod paydo bo'ldi.

O'n million raqamlarini tartiblashtirish uchun A kompyuterga kerak bo'ladi:

$$\frac{2 * (10^7)^2}{10^{10}} = 20000$$

B kompyuterga kerak bo'ladi

$$\frac{50 * 10^7 \lg 10^7}{10^7} \approx 1163$$

Ko'rib turganingizdek, kod bilan foydalanish, ish vaqti sekin ko'tarilganda,yomon komilyator bilan ham eng sekin kompyuterda ham17 marta kam vaqt talab qiladi.

Qo'shish usuli joylashtirish usulidan samaraliroq ekanligini quyida keltirilgan jadlal ma'lumotlarini tahlili orqali keltiramiz.

Kompyuterlar	Saralanadigan sonlar soni	Saralovchi algoritm	Talab qilinadigan vaqt
A(tez ishlovchi 1sekundda 10mlrd amal bajaradi)	10 mlnta (taqriban 80 mb)	Joylashtirish usuli (tajribali dasturchi tomonidan yaratilgan algoritm saralash uchun $2n^2$ amal bajariladi)	$\frac{2 * (10^7)^2 \ buyruqlar}{10^{10} \ buyruq/sec}$ $=20000 \ sec$ *(5,5 soatdan ko'proq)*

B(sekin ishlovch-Isekund da 10 mln amal bajaradi)		Qo‘shi sh usuli (o‘rta darajali dasturchi tomonidan yaratilgan algoritm saralash uchun 50nlgn amal bajariladi))	$\frac{50 * 10^7 \ lg10^7}{10^7} \approx 1163 \ sekund$ (20 min *dan kam*)

2. Algoritmlar va boshqa texnologiyalar

Yuqoridagi misol shuni ko‘rsatadiki, kompyuter apparat kabi algoritmlarni ham, texnologiya sifatida hisobga olinishimiz kerak.

Umumiy tizim ish faoliyatini algoritm samaradorligiga ham bog‘liq, va apparat kuchiga ham. Algoritm rivojlantirish sohasida jadal rivojlantirish bo‘lyapti, boshqa kompyuter texnologiyalaridek.

Savol tug‘iladi, algoritmlar shunchalik muhimi, zamonaviy kompyuterlarda ishlaydigan bo‘lsin, agar shunday kabi yuqori texnologiyalar boshqa sohalarda ulkan yutuqlarga erishilgan bo‘lsa

- zamonaviy kompyuter mimarileri va ularning ishlab chiqarish texnologiyalari;
- osonlik bilan erishish, intuitiv grafik foydalanuvchi interfeysi (GUI);
- Ob'ektga yo‘naltirilgan tizimlar;
- Integratsiyalashgan veb texnologiyasi;
- tezroq tarmoqlari, simli va simsiz.

Misol uchun, bir joydan boshqasiga olish uchun qanday belgilaydigan Web xizmat. Uni amalga oshirish bir yuqori samarali apparat, grafik foydalanuvchi interfeysi, bir global tarmoq va, ehtimol, bir ob'ekt yo‘naltirilgan yondashuv yotadi.

Bundan tashqari, bunday yo‘nalishlarini topish kabi bir berilgan veb-xizmati tomonidan amalga muayyan operatsiyalar uchun zarur algoritmlarni foydalanish, ko‘rish va enterpolasyon manzilini, xaritalar bilan foydalaniladi. Bundan tashqari, dastur,yuqori saviyada algoritmik mazmunini talab qilmaydi, kuchli algoritmlarga bog‘liq. Bu dastur ishlash apparat ishiga bog‘liq ekanligi ma'lum, va amaliy rivojlanishida turli algoritmlardan foydalaniladi.

Biz hammamiz bilamizki, ilova yaqindan grafik foydalanuvchi interfeysi bilan bog‘liq, va har qanday grafik foydalanuvchi interfeysini ishlab chiqish uchun talab algoritmlari kerak bo‘ladi. Tarmoq ustida ishlaydigan ilovalarni eslatib o‘tamiz.

Ular faoliyat olib borishlari uchun, algoritmlarga asoslanga yo'nalishni olib borishlari kerak bo'ladi. Eng keng tarqalgan dasturlar tilda tuziladi, mashinadan farqli. Ularning kodi turli kompilyator va interpretatorlar bilan ishlov beriladi, turli algoritmlardan keng foydalanadi. Bundan tashqari, kompyuterlar kuchini doimiy o'sishi, ular tobora murakkab vazifalarni hal qilish uchun qo'llaniladi. Biz muammoni murakkabligini ortishimiz bilan, ikki saralash usullari qiyosiy tahlili misolida ko'rib turganimizdek eng muhim farqlar algoritmlarini samaradorligini ko'rinadi oshirilmoqda.Asosiy algoritmlar va ularni rivojlantirish usullari-asosiy xususiyatlatdan biri.Zamonaviy kompyuter texnologiyalari bilan, ayrim vazifalarni algoritmlarni bilmagan holda ham qilinishi mumkin, lekin bu sohada kop narsaga erishish mumkin.

Mashqlar

1.2.1 Dastur darajasida zarur bo'lgan algoritmik content dasturini misol qilib keltiring va bu algoritmlarni funktsiyasini muhokama qiling

1.2.2 Deylik, bitta mashinada ikkita saralash algoritmni qiyosiy tahlil amalga oshirilmoqda. N elementlarni joylashtirish saralashi uchun $8n^2$ kerak bo'ladi, birlashtirish saralashi uchun 64n lg n qadamlar kerak bo'ldi. Joylashtirish saralashi birlashtirish saralashidan qiymati oshsa, n ni qiymati qancha bo'lishi kerak?

1.2.3 Ikkita algoritm bitta mashinada amalda bo'lsa, n algoritmni qaysi minimal qiymati, ish vaqti $100n_2$ formula bilan aniqlansa, ish vaqti 2^n formula bilan aniqlangan qaysi biri tezroq ishlaydi. Kimning yugurib vaqt 100p2 bilan belgilanadi, uning ish vaqti, ham algoritmlar Shu mashina amalga bo'lsa, sifatida ifodalangan bir algoritm tezroq n algoritm qaysi minimal qiymati da?

Masala

1.1. Algoritmlarni ish vaqtini solishtirish

Quyida bir jadval bo'lib, satrlari turli vazifalarga mos f(n), ustunlaru esa- vaqt qiymatiga t.

N ni maksimal qiymatlari bilar jadvalni toldiring, masala t vaqti bilan yechilishi mumkin,agar masalani yechish uchun algoritmni ish vaqti f(n)mikro sekundga teng bo'lsa.

	1 sekund	1 minut	1 soat	1 kun	1 month	1 yil	1 asr
$\lg n$							
$\sqrt{n}$							
n							
$n\lg n$							
n^2							
n^2							
n^2							

Suppose computers were infinitely fast and computer memory was free. Would you have any reason to study algorithms? The answer is yes, if for no other reason than that you would still like to demonstrate that your solution method terminates and does so with the correct answer.

If computers were infinitely fast, any correct method for solving a problem would do. You would probably want your implementation to be within the bounds of good software engineering practice (for example, your implementation should be well designed and documented), but you would most often use whichever method was the easiest to implement.

Of course, computers may be fast, but they are not infinitely fast. And memory may be inexpensive, but it is not free. Computing time is therefore a bounded resource, and so is space in memory. You should use these resources wisely, and algorithms that are efficient in terms of time or space will help you do so.

Efficiency

Different algorithms devised to solve the same problem often differ dramatically in their efficiency. These differences can be much more significant than differences due to hardware and software.

As an example, in Chapter 2, we will see two algorithms for sorting. The first, known as ***insertion sort***, takes time roughly equal to $c_1 n^2$ to sort n items, where c_1 is a constant that does not depend on n. That is, it takes time roughly proportional to n^2. The second, ***merge sort***, takes time roughly equal to $c_2 n \lg n$, where $\lg n$ stands for $\log_2 n$ and c_2 is another constant that also does not depend on n. Insertion sort typically has a smaller constant factor than merge sort, so that $c_1 < c_2$. We shall see that the constant factors can have far less of an impact on the running time than the dependence on the input size n. Let's write insertion sort's running time as $c_1 n \cdot n$ and merge sort's running time as $c_2 n \cdot \lg n$. Then we see that where insertion sort has a factor of n in its running time, merge sort has a factor of $\lg n$, which is much smaller. (For example, when $n = 1000$, $\lg n$ is approximately 10, and when n equals one million, $\lg n$ is approximately only 20.) Although insertion sort usually runs faster than merge sort for small input sizes, once the input size n becomes large enough, merge sort's advantage of $\lg n$ vs. n will more than compensate for the difference in constant factors. No matter how much smaller c_1 is than c_2, there will always be a crossover point beyond which merge sort is faster.

For a concrete example, let us pit a faster computer (computer A) running insertion sort against a slower computer (computer B) running merge sort. They each must sort an array of 10 million numbers. (Although 10 million numbers might seem like a lot, if the numbers are eight-byte integers, then the input occupies about 80 megabytes, which fits in the memory of even an inexpensive laptop computer many times over.) Suppose that computer A executes 10 billion instructions per second (faster than any single sequential computer at the time of this writing) and computer B executes only 10 million instructions per second, so that computer A is 1000 times faster than computer B in raw computing power. To make the difference even more dramatic, suppose that the world's craftiest programmer codes insertion sort in machine language for computer A, and the resulting code requires $2n^2$ instructions to sort n numbers. Suppose further that just an average programmer implements merge sort, using a high-level language with an inefficient compiler, with the resulting code taking $50n \lg n$ instructions. To sort 10 million numbers, computer A takes

$$\frac{2 \cdot (10^7)^2 \text{ instructions}}{10^{10} \text{ instructions/second}} = 20{,}000 \text{ seconds (more than 5.5 hours)},$$

while computer B takes

$$\frac{50 \cdot 10^7 \lg 10^7 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 1163 \text{ seconds (less than 20 minutes)} .$$

By using an algorithm whose running time grows more slowly, even with a poor compiler, computer B runs more than 17 times faster than computer A! The advantage of merge sort is even more pronounced when we sort 100 million numbers: where insertion sort takes more than 23 days, merge sort takes under four hours. In general, as the problem size increases, so does the relative advantage of merge sort.

Algorithms and other technologies

The example above shows that we should consider algorithms, like computer hardware, as a *technology*. Total system performance depends on choosing efficient algorithms as much as on choosing fast hardware. Just as rapid advances are being made in other computer technologies, they are being made in algorithms as well.

You might wonder whether algorithms are truly that important on contemporary computers in light of other advanced technologies, such as

- advanced computer architectures and fabrication technologies,
- easy-to-use, intuitive, graphical user interfaces (GUIs),
- object-oriented systems,
- integrated Web technologies, and
- fast networking, both wired and wireless.

The answer is yes. Although some applications do not explicitly require algorithmic content at the application level (such as some simple, Web-based applications), many do. For example, consider a Web-based service that determines how to travel from one location to another. Its implementation would rely on fast hardware, a graphical user interface, wide-area networking, and also possibly on object orientation. However, it would also require algorithms for certain operations, such as finding routes (probably using a shortest-path algorithm), rendering maps, and interpolating addresses.

Moreover, even an application that does not require algorithmic content at the application level relies heavily upon algorithms. Does the application rely on fast hardware? The hardware design used algorithms. Does the application rely on graphical user interfaces? The design of any GUI relies on algorithms. Does the application rely on networking? Routing in networks relies heavily on algorithms. Was the application written in a language other than machine code? Then it was processed by a compiler, interpreter, or assembler, all of which make extensive use

of algorithms. Algorithms are at the core of most technologies used in contemporary computers.

Furthermore, with the ever-increasing capacities of computers, we use them to solve larger problems than ever before. As we saw in the above comparison between insertion sort and merge sort, it is at larger problem sizes that the differences in efficiency between algorithms become particularly prominent.

Having a solid base of algorithmic knowledge and technique is one characteristic that separates the truly skilled programmers from the novices. With modern computing technology, you can accomplish some tasks without knowing much about algorithms, but with a good background in algorithms, you can do much, much more.

Exercises

1.2-1
Give an example of an application that requires algorithmic content at the application level, and discuss the function of the algorithms involved.

1.2-2
Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

1.2-3
What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

[1] Thomas H. Cormen va b. Intruduction to algorithms. Massachusetts Institute of Technology. London 2009. (11-13pp)

3. Tarixiy ma'lumotlar.

Bizgacha yetib kelgan intuitiv ma'nodagi algoritm eramizdan avvalgi III- asrda Yevklid tomonidan taklif qilingan. Ushbu algoritm juda mashhur bo'lib, XX-asr boshlarigacha «algoritm» so'zining o'zi «Yevklid algoritmi» ma'nosida ishlatilib kelindi. Boshqa matematik masalalarni bosqichli yechishni tasvirlash uchun esa «usul» so'zidan foydalanilgan.

Zamonaviy algoritmlar nazariyasi rivojidagi boshlang'ich nuqta deb, nemis matematigi Kurt Gyodelning ilmiy ishini ko'rsatib o'tish mumkin (1931 y. Simvolik mantiqlarning to'lamasligi to'g'risidagi teorema). Ushbu ishda ba'zi matematik muammolarni qaysidir sinfga taalluqli algoritmlar yordamida hal etib bo'lmasligi ko'rsatib berilgan.

1936 yilda Algoritmlar nazariyasi bo'yicha birinchi fundamental ilmiy ishlar bir-biridan alohida tarzda Alan Tyuring, Aloiz Chyorch va Emil Postlar e'lon qildilar. Ular tomonidan taklif etilgan TM, Post mashinasi va Chyorchning lyamda-hisoblanuvchanlik usuli algoritm formalizmining ekvivalent shakllaridir. Ular tomonidan taklif etilgan tezislar algoritm intuitiv tushanchasi va formal tizimlarning ekvivalentligini ta'kidlab berdi. Algoritmik yechimsiz muammolarning formulirovkasi va isboti ushbu ishlarning muhim natijasi bo'ldi.

1950 - yillarda Algoritmlar nazariyasi rivojlanishiga rus matematiklari Kolmogorov va Markovlari z hissalarini qo'shdilar. 60-70-yillarga kelib Algoritmlar nazariyasi fanida quyidagi mustaqil yo'nalishlar ajralib chiqdi:

- Klassik algoritmlar nazariyasi (formal tillar terminlarida masalalarni ifodalash, yechimli masala tushunchasi, 1965 yilda Edmonds tomonidan ta'riflangan P=NP muammosi, NP to'liq masalalar sinfining ochilishi va tekshirilishi);
- Algoritmlarning asimptotik tahlili nazariyasi (asimptotik baholash usullari, algoritmlarning murakkabligi, algoritmlarni baholash kriteriylari va h.k.). Ushbu yo'nalish rivojiga Knut, Axo, Xopkroft, Ulman, Karp kabi olimlar o'z hissalarini qo'shdilar;
- hisoblash algoritmlarining amaliy tahlili nazariyasi (algoritmlarning mehnattalabligi oshkor funksiyasini topish, funksiyalarning chegaraviy tahlili, ratsional algoritmlarni tanlash metodikasi). Ushbu yo'nalish rivojlanishiga sabab bo'lgan ilmiy ish D.Knutning "Исскуство программирования для ЭВМ" kitobidan iborat.

4. Algoritmlar nazariyasi fanining maqsadi va vazifalari.

Algoritmlar nazariyasi – Informatika va tadbiqiy matematikaning fundamental qismiga oid fan bo'lib, uning davomi bevosita samarali dasturlarni tuzish, sonli usullar, mukammallashtirish usullari va ob'ektga yo'naltirilgan dasturlash sohalarida o'z nazariy va amaliy tadbiqini topadi.

Fanning maqsadi – talabalarda umumiy(intuitiv) va formal ma'nodagi algoritm haqidagi tushunchalarni ajrata olish, muayyan masalani yechish uchun algoritm mavjudligi haqida tushunchalarni shakllantirish,

masalani yechish uchun tatbiq etilishi mumkin bo'lgan algoritmlar orasida eng samaralisini ajratib olish, yaratilgan yoki mavjud algoritmlarni murakkablik ko'rsatkichlarini baholash kabi bir qator nazariy va amaliy muammolar bo'yicha bilim va ko'nikmalarni hosil qilishdan iborat.

Fanning vazifasi – algoritmning formal ta'riflari asosida yechimga ega masalalarning natijaviyligini o'rnata bilishga erishish. Muayyan algoritmlarning murakkablik funksiyasi qanday bo'lishini anglashga o'rgatish va amalda sinash ko'nikmasini hosil qilish.

Algoritmlar nazariyasi turli yo'nalishlarining yutuqlarini umumlashtirib, uning maqsadi va vazifalarini quyidagicha ham ko'rsatib o'tish mumkin:

- Algoritm tushunchasini formallashtirish va formal algoritmik tizimlarni tekshirish;
- Bir qator masalalarning algoritmik yechimsizligini formal isbotlash;
- Masalalar klassifikasiyasi, murakkablik sinflarini aniqlash va tekshirish;
- Algoritmlar murakkabligining asimptotik tahlili;
- Rekursiv algoritmlarni tekshirish va tahlil qilish;
- Algoritmlar qiyosiy tahlili uchun mehnattalablik oshkor funksiyasini topish;
- Algoritmlar sifatini qiyosiy baholash kriteriylarini ishlab chiqish.

5. Algoritmlar nazariyasining nazariy va amaliy ahamiyati.

Algoritmlar nazariyasidan olingan nazariy natijalardan amalda ancha keng foydalanilmoqda. Bunda ikki aspektni alohida ko'rsatib o'tish mumkin:

Nazariy aspekt: qandaydir masalani tekshirish natijasida "Ushbu masala prinsipial jihatdan algoritmik yechimlimi? - degan savolga javob berish imkoniyati mavjud. Algoritmik yechimsiz masalalar TM to'xtashi masalasiga olib kelinishi mumkin. Algoritmik yechimli masalalar uchun esa, ushbu masalaning NP to'liq masalalar sinfiga mansubligi muhim ikkinchi nazariy savol bo'lib hisoblanadi.

Amaliy aspekt: Algoritmlar nazariyasi usullari quyidagi vazifalarni bajarishga imkon beradi:

- Berilgan masalani yechish algoritmlari to'plamidan eng rasional algoritmni tanlash;
- Murakkab masalalarni yechish algoritmlarini vaqt jihatidan baholash;
- Kriptografik metodlar uchun muhim bo'lgan masala yechimini ma'lum vaqt oralig'ida olib bo'lmasligini ishonchli baholash;
- Amaliy tahlil asosida axborotlarni qayta ishlash sohasidagi masalalarni yechish effektiv algoritmlarini ishlab chiqish va rivojlantirish.

Nazorat savollari:

1. Algoritmlar nazariyasi faniga hissa qo'shgan olimlardan kimlarni bilasiz?
2. Algoritmlar nazariyasi fanining maqsadlari nimadan iborat?
3. Algoritmlar nazariyasi fanining vazifalari nimalardan iborat?
4. Algoritmlar nazariyasi fani qaysi yo'nalishlar bo'yicha rivojlanib keldi?
5. Algoritmlar nazariyasi fani yutuqlarining nazariy va amaliy ahamiyati nimadan iborat?

Algoritmlar tahlili.
Algoritmlarni ishlab chiqish metodlari.

MA'RUZA №3
MAVZU: ALGORITMLAR TAHLILI.

REJA:

1. Algoritm sifatini baholashning mezonlari.
2. Algoritmlarni tahlil qilish usullari.
3. Algoritmni to'g'riligini tekshirish.

Dastur to'g'riligini isbotlashning eng keng tarqalgan turi – bu uni testlardan o'tkazishdir.

Algoritmni tekshirishda nazoratchi boshlang'ich ma'lumotlarni majmui algoritmik test deb nomlanadi.

To'g'ri deb shunday algoritmga aytiladiki, u masalaning qo'yilishida talab qilinadigan natijani har qanday ruxsat etilgan boshlang'ich ma'lumotlar bilan ham shakllantirib biladi. Odatda, dastur bergan natijalar ma'lum bo'lgan yoki qo'lda hisoblangan ma'lumotlar bilan taqqoslanadi, va ular to'g'riligi aniqlansa dastur to'g'ri ishlaydi degan hulosaga kelish mumkin. Ammo bu usul bilan foydalanuvchini hamma shubhalardan xalos qilib bo'lmaydi, ya'ni dastur ishlamaydigan hamma holatlarni hisobga olib bo'lmaydi.

Gudman va Xidetniyemi [2] lar tomonidan algoritm to'g'riligini isbotlash uchun quyidagi uslubiyat taklif qilingan.

Algoritm 0 dan *m* gacha bo'lgan qadamlar ketma-ketligi ko'rinishida tavsiflangan deb tahmin qilaylik.Har bir qadam uchun qandaydir asoslanishni taklif etamiz. Xususan, qadamdan oldin va keyin ishlaydigan shartlar haqida lemma kerak bo'lishi mumkin. Shu bilan birgalikda, algoritm chekliligining isbotini ham taklif etamiz, va hamma ruxsat etilgan kiritish ma'lumotlarini tekshirib, hamma mumkin bo'lgan chiqarish ma'lumotlarni olamiz. Algoritmni to'g'riligi bilan samaradorligi o'rtasida hech qanday aloqa yo'qligini ta'kidlab o'tamiz.Aslida hamma talablarga bir xil yahshi javob beradigan algoritm kamdan-kam ishlab chiqiladi.

Algoritmni amalga oshirish

Algoritmni amalga oshirish deganda, EHM uchun dasturni yozish deb tushuniladi. Buning uchun quyidagi savollarga javob berish kerak:

5.1. Asosiy o'zgaruvchilarni aniqlash.
5.2. O'zgaruvchilarning turlarini aniqlash.
5.3. Nechta massiv yoki fayllar va qanday kattalikda ular kerak bo'ladi?
5.4. Bog'lanilgan ro'yhatlardan foydalanish ma'nolimi?
5.5. Qanday dasturiy qismlar kerak bo'lishi mumkin (tayyor bo'lsa ham)?

5.6. Qaysi dasturlash tilini tanlash?

Dastur yozish yoki tuzishning hilma-hil usillari va uslublari mavjud.

Algoritmni va uning murakkabligini tahlil qilish

Algoritmni tahlil qilishdan maqsad – algoritmga ma'lumotlarni aniq muvaffaqiyatli qayta ishlash uchun kerak bo'ladigan xotira hajmi va ishlash vaqtining baholari va chegaralarini olish. Bir masalani yechadigan ikki algoritmni taqqoslash uchun qandaydirsonli mezon topish kerak.

Faraz qilaylik, A – qandaydir bir turkumdagi masalalarni yechadigan algoritm, n – esa shu turkumdagi alohida bir masalaning kattaligi.Umumiy holda, n – oddiy skalyar yoki massiv yoki kiritiladigan ketma – ketlikning uzunligi bo'lishi mumkin. $f_A(n)$ - n kattalikdagi ixtiyoriy masalani yechadigan algoritm A bajarish kerak bo'lgan asosiy amallarni (qo'shish, ayirish, taqqoslash,…) yuqori chegarasini beradigan ishchi funksiya. Algoritmningsifatini baholash uchun quyidagi mezonni ishlatamiz.

Agar $f_A(n)$ o'sish tartibi n dan bog'liq bo'lgan polinomdan katta bo'lmasa, A algoritm polinomial deb aytiladi, aks holda algoritm A eksponensial hisoblanadi.

Shular bilan birgalikda tahlil jarayonida ko'p matematik fanlarda standart bo'lgan iboralar ishlatiladi.

$f_A(n)$ funksiya O[g(n)] deb belgilanadi, va $\lim_{n\to\infty}\frac{f(n)}{g(n)} = const \neq 0$ bo'lganda, uni tartibi katta n lar uchun $g(n)$ deb qabul qilinadi. Demak $f(n)=O[g(n)]$.

$f_A(n)$ funksiyasi o[z(n)] deb katta n lar uchun belgilanadi, va unda $\lim_{n\to\infty}\frac{h(n)}{z(n)} = 0$ sharti bajariladi.

Bu begilar "katta O" va "kichik o" deb nomlanadi. Agar $f(n)=O[g(n)]$ bo'lsa, ikkala funksiya ham $n\to\infty$ bo'lganda bir xil tezlikda o'sadi.

Agar $f(n)=O[g(n)]$ bo'lsa,unda g(n), f(n) nisbatan ancha tez o'sadi.

Demak, $P_k(n)$- qandaydir n o'zgaruvchidan bog'liq va k darajadagi polinom uchun $f_A(n) = O[P_k(n)]$ yoki $f_A(n) = oP_k(n)$ bo'lganda algoritm polynomial hisoblanadi, aks holda algoritm eksponensial.

Eksponensial algoritm yahshi ishlamaydigan deb hisoblanadi.Agar algoritmlar eksponensial bo'lsa, ular orasida eng samaralisini topish kerak, n kattalikdagi masalani $O(2^n)$ qadamda yechadigan algoritm $O(n!)$ yoki $O(n^n)$ qadamda masalani yechadigan algoritmdan afzalroq.

Dasturni tekshirish

Biz dasturni har bir qismini tekshiradigan kirituvchi ma'lumotlar to'plamini tanlashimiz kerak.Ko'p murakkab algoritmlarni matematik tomondan tadqiq qilish yoki juda qiyin yoki mumkin emas. Bunday holatlarda algoritmni faoliyat jarayonida va qiyinligi bo'yicha tekshiradi. Bundan tashqari dasturlarni hisoblash imkoniyatlarini aniqlash uchun ham testlash maqsadga muvofiq.Ko'p

dasturlar qandaydir kiritiladigan ma'lumotlar bilan yahshi ishlasa, boshqalari bilan yomon ishlaydi."Yahshi" lardan "yomon" larga o'tish "mayin" bo'lish kerak.Testlash uchun ma'lumotlar dasturning qiyinligiga, mavjud vaqt resurslariga, kiritish-chiqarishsoniga bog'liq holda tanlanadi. Bu yerda analitik va eksperimental tahlil bir-birini to'ldiradi.

Hujjatlashtirish

O'zingiz yozmagan dastur kodini o'qish juda qiyin.Bu muammoni hujjatlashtirish yordamida yechsa bo'ladi. Hujjatlashtirish o'z ichiga hamma yordamchi ma'lumotlarni oladi va dasturda nima bajarilishini tushuntirib beradi, xususan, blok-sxemalardagi boshqarishni uzatish, berilganlarni kiritish-chiqarish shaklini batafsil tavsif qilish, siklning parametrlari, yordamchi local va global proseduralarni bajarilishi va boshqalar.

Hujjatlashtirishning eng asosiy qoidasi bu "boshqalar yozgan dasturlarni qanday ko'rishni istasangiz, o'zingiz ham dasturni shunday ko'rinishda rasmiylashtiring".

Masalalar yechish.

1 - misol. *Berilgan to'rt xonali butun sonning raqamlari ko'paytmasini toping.*

Test

Test tartibi	Tekshirish	Son	Natija
1	Musbat son	2314	P = 24
2	Manfiy son	-1245	P = 40

Algoritmi:

```
alg Butun_son (but Num, P)
arg Num
natija P
boshlbutun i, j, k, l
 Num := abs(Num)
 i := Num div 1000
j := ((Num div 100) mod 10)
k := ((Num div 10) mod 10)
 l := Num mod 10
 P := i * j * k * l;
Tamom
```

Turbo Pascaldagi dasturi:

```
Program Farrux;
Var Number, i, j, k, l, P : Integer;
BEGIN
 ReadLn(Number);  Number:=Abs(Number);
  i := Number div 1000; Write(i:3);
j := Number div 100 mod 10; Write(j:3);
```

```
k := Number div 10 mod 10; Write(k:3);
l := Number mod 10; WriteLn(l:3);
P := i * j * k * l ;  WriteLn( P);
 ReadLn
END.
```

2 - misol. Butun qiymatli A(N, M) matritsa berilgan. Agar matritsa satrining hech bo'lmaganda biror elementi manfiy bo'lsa, u holda bu satrning barcha elementlarini
nollar bilan almashtiring

Test

Berilgan		Natija
N	A matritsa	A matritsa
3	$\begin{pmatrix} 1 & -2 & 1 \\ 1 & 2 & 1 \\ -1 & 2 & -2 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix}$

Algoritmi

```
alg Modifikasiya(but N, haq jad A[1:N, 1:N])
 boshl but i, j, lit Flag
 kiritish N
 sb  iuchun 1 dan N gacha
  sbj uchun 1 dan N gacha
    kiritishA[i,j]
  so
 so
 sbi uchun 1 dan N gacha
  j := 1; Flag := "Yuk"
  sb toki (j<=N) va (Flag = "Yo'q")
    agar A[i, j]<0  u holda Flag := "Ha"
     aks holda j:=j+1
    hal bo'ldi
  so
  agar Flag = "Ha"u holda
sbj uchun 1 dan N gacha A[i, j]:=0
     so
  hal bo'ldi
 so
 sbi uchun 1 dan N gacha
  sbj uchun 1 dan N gacha
```

chiqarishA[i,j]
so
so
tamom.

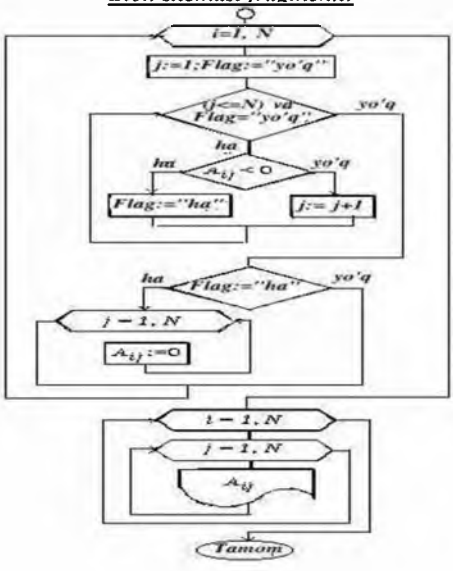
Algoritmning bajarilishi

Tekshirilayotgan shartning belgilanishi:

(j<=N) va (Flag = "Yo'q")=> (1)

i	Flag	j	(1)	A[i,j]<0	Flag="Ha"	A[i,j]
1	"Yo'q"	1	+	-	+	
	"Ha"	2	+	+		A[1,1]=0
		1	-(so)			A[1,2]=0
		2				A[1,3]=0
		3				
2	"Yo'q"	1	+	-	-	
		2	+	-		
		3	+	-		
		4	-(so)			
3	"Yo'q"	1	+	+	+	A[3,1]=0
	"Ha"	1	-(so)			A[3,2]=0
		2				A[3,3]=0
		3				

Blok-sxemasi fragmenti:



Turbo Pascaldagi dasturi:

```
Program Modify;
 Var A      : Array[1..10, 1..10] of Real;
    N, i, j : Integer;
Procedure InputOutput;
 Begin
  ReadLn(N);
   For i := 1 to N do
    For j := 1 to N do
     begin Write('A[' , i , ',' , j , '] = ');
         ReadLn(A[i, j])
     end;
   For i := 1 to N do
    begin
      For j := 1 to N do Write(A[i, j] : 5 : 1);
      WriteLn
    end;
 End; { of InputOutput }
{-----------------------------------------}
Procedure Line(Var i : Integer);
 Var Flag : Boolean;
 Begin
  j := 1; Flag := FALSE;
   While (j<=N) and not Flag do
    If A[i, j]<0 then Flag:=TRUE else j:=j+1;
   If Flag then
    For j := 1 to N do A[i, j] := 0
End;
{-----------------------------------------}
Procedure OutRes;
 Begin
  WriteLn(' Natija- Matritsa:'); WriteLn;
   For i := 1 to N do
    begin
      For j := 1 to N do Write(A[i, j]:5:1);
      WriteLn
    end; ReadLn
 End; { of OutRes }
BEGIN
  InputOutput;
  For i := 1 to N do Line(i);
  OutRes;
  END.
```

MA'RUZA №4

MAVZU: ALGORITMNI ISHLAB CHIQISH METODLARI.

REJA:

1. Sxemalar va yangi algoritmlarni paydo qilishning usullari.
2. Algoritmik yechimga ega bo'lmagan masalalar.
3. O'z-o'ziga qo'llanuvchanlik muammosi.
4. Tyuring mashinasining o'z-o'ziga qo'llanuvchanligi.

Kalit so'zlar: Algoritmik yechimsizlik, diofant tenglama, hisoblanuvchi bo'lmagan funksiya, qo'llanuvchanlik, o'z-o'ziga qo'llanuvchanlik, kirish so'zi, chiqish so'zi.

1. Sxemalar va yangi algoritmlarni paydo qilishning usullari.

Algoritmlarni yaratish ijobiy ish, shuning uchun ixtiyoriy zarur algoritmlarni tuzish imkonini beradigan bir umumiy usul mavjud emas. Lekin algoritmlarni ishlab chiqishni asoslangan oddiy sxemalarini beradigan ko'pgina algoritmlashtirish nazariyalari bor. Bunday sxemalar va yangi algoritmlarni paydo qilishning o'rtasida bog'liqlik kuzatiladi. Tez uchraydigan va ko'p foydalaniladigan usullarni quyidagicha ajratib olish mumkin:

1. **Algoritmlarni konstruksiyalash.** Bu usulda yangi algoritm mavjud algoritmlardan tarkibiy qismlar sifatida foydalanib, bir-biriga moslab bir butunlik hosil qilish yo'li bilan ishlab chiqiladi.
2. **Algoritmlarni ekvivalent qayta ishlash.** Ikki algoritm ekvivalent hisoblanishi uchun quyidagi shartlar bajarilish kerak:
   - Bittasi uchun mumkin bo'lgan dastlabki berilganlar varianti, ikkinchisi uchun ham mumkin bo'lishi kerak.
   - Bir algoritmni qandaydir dastlabki ma'lumotga qo'llanilishi, ikkinchi algoritmni ham shu berilganga qo'llanilishiga kafolat beradi.
   - Bir xil dastlabki berilgan ma'lumot uchun ikkala algoritm ham bir xil natija berishi. Lekin bu algoritmni ikki xil shakllarini ekvivalent deb nomlash noto'g'ridir.

Shunday qilib, algoritmni ekvivalent qayta ishlash deb, natijada dastlabki algoritmga ekvivalent algoritmni paydo qiladigan o'zgartirilishlarga aytiladi.

Misol tariqasida, algoritmni bir tildan boshqa tilga o'tkazishni keltirish mumkin. Shu bilan birgalikda algoritmni ekvivalent qayta ishlash usuli bilan keskin o'zgartirish mumkin, lekin bu holda asosiy e'tiborni dastlabki algoritmga nisbatan yahshi algoritmni yaratishga berish kerak.

3. **Toraytiruvchi o'zgartirishlar.** Bunday o'zgartirishlar natijasida dastlabki algoritmlar yechish kerak bo'lgan masalalarning xususiy holati yechimi algoritmlari ishlab chiqiladi. Odatda, bu usulda ekvivalent qayta ishlash jarayonida algoritmni ixchamlashtirish maqsaddida foydalaniladi.
4. **Formal usulni matematikaga bog'liq bo'lmagan muammoga qo'llash.** Buyerda matematik muammo matematik ko'rinishga o'tkazilib, uning algoritmini ishlab chiqishga uriniladi. Agar o'xshash matematik masala yechimining algoritmi ma'lum bo'lsa, undan foydalaniladi.

2. Algoritmik yechimga ega bo'lmagan masalalar.

Algoritmlar nazariyasida shunday masalalar mavjudki, ushbularni yechish algoritmlari mavjud emas. Bunday masalalar ***algoritmik yechimsiz*** deb ataladi. Odatda "*yangi*" masalalarning algoritmik yechimsizligi ularni oldindan ma'lum algoritmik yechimsiz masalalarga keltirish yo'li bilan isbotlanadi. Shu bilan birga yangi masalaning yechimi mavjud bo'lganda oldindan yechimsiz deb hisoblangan masalani ham yechish mumkinligi isbotlanadi. Bunday masalalar qatoriga ***o'z-o'ziga qo'llanuvchanlik*** muammosi misol bo'ladi.

Algoritmga aniq ta'rif berilganidan so'ng berilgan ommaviy muammolar algoritmik yechimga ega bo'lish yoki bo'lmaslik masalasini hal etish imkoniyatlari paydo bo'ldi. Algoritmik yechimga ega bo'lmagan masalalar na'munalarini ko'rib chiqamiz .

1936-yili A.Chyorch tomonidan predikatlar hisobi uchun formulalarning umumqiymatli bo'lish yoki bo'lmasligini hal qiladigan algoritm mavjud emasligi isbotlandi.

1-***ta'rif.*** *Biror bir alifboning so'zlar to'plami o'zining chekli sondagi o'rniga qo'yish qoidalari bilan birgalikda assotsiativ hisob deyiladi.*

Assotsiativ hisobning ixtiyoriy ikkita so'zi uchun bu ikkita so'zning teng kuchli bo'lish-bo'lmaslik masalasi assotsiativ hisobda so'zlarning *ekvivalentlik* muammosi deyiladi.

Bu masala 1911-yilda e'lon qilingan. 1946–47-yillarda rus matematigi A.A.Markov va amerikalik matematik E.Postlar ekvivalentlik muammosi algoritmik yechimga ega emasligini hal etganlar.

1955-yilda rus matematigi P.S.Novikov gruppalar nazariyasida so'zlar ekvivalentligi muammosi algoritmik yechimga ega emasligini isbotladi.

1900-yilda matematiklarning Parijda bo'lib o'tgan ikkinchi halqaro kongressida yechilishi qiyin bo'lgan 23 ta matematik muammolar e'lon qilindi. Shu muammolarning o'ninchisida har qanday butun koeffitsientli n ta o'zgaruvchili ko'phad butun ildizlarga ega bo'lish, bo'lmasligini aniqlaydigan algoritm bor yoki yo'qligini aniqlashdan iborat edi. Bunday ko'phadlarga quyidagilar misol bo'ladi :

$f(x, y, z) = x^2 + y^2 + z^2 - 2xyz$,

$f(x) = 5x^3 - x^2 + x + 15$.

Hususiy holda butun koeffitsientli bir noma'lumli

$f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n \quad (a_0 \neq 0)$

ko'rinishdagi n darajali ko'phadning butun yechimlarini topish algoritmi mavjud ekanligi ma'lum.

1968-yili yuqorida keltirilgan masala umumiy holda algoritmik yechimga ega emasligi Yu.Matiyasevich tomonidan isbot qilindi.

3. O'z-o'ziga qo'llanuvchanlik muammosi.

Tyuring mashinasi haqida gapirganda, ixtiyoriy Tyuring mashinasi sxemasini kodlangan holda tasmaga yozish mumkinligini bilamiz. Xuddi shuningdek, ixtiyoriy Markovning normal algoritmining o'rinlashtirish formulalarini ajratish uchun biror belgidan foydalanib kodlash mumkin. U holda Normal algoritmning o'zi so'zga aylanadi va ixtiyoriy Normal algoritm uchun KIRISH so'zi sifatida qo'llanilishi mumkin. Xususiy holda Normal algoritm o'z-o'ziga KIRISH so'zi bo'ladi.

Barcha algoritmlar ikki sinfga bo'linadi: o'z-o'ziga qo'llanuvchan va qo'llanilmas:

O'z-o'ziga qo'llanuvchan algoritmlar deb, o'zining ifodasi ustida ishlab, ertami-kechmi to'xtaydigan algoritmlarga aytiladi. Agar algoritm ishi cheksiz takrorlanuvchi bo'lsa, bunday algoritmlar o'z-o'ziga qo'llnailmas deyiladi.

Shunday qilib, haqli savol tug'iladi: Qanday qilib u yoki bu algoritmning o'z-o'ziga qo'llanuvchanligini aniqlash mumkin? Ya'ni, ixtiyoriy algoritm uchun yuqoridagi savolga javob beruvchi umumiy algoritm topilishi kerak.

3. Tyuring mashinasining o'z-o'ziga qo'llanuvchanligi.

Ishni hech qaysi Tyuring mashinasi yordamida hisoblab bo'lmaydigan funksiya qurishdan boshlaymiz.

Hisoblanuvchi bo'lmagan funksiyaga misol. Buning uchun foydalanish mumkin bo'lgan barcha Tyuring mashinalarini ifoda etamiz: Tyuring mashinasidagi ichki holatlarni cheksiz q_0, q_1, q_2, ... q_s, ... lar bilan belgilaymiz. Ushbu mashinalar majmui alfavitlari a_0, a_1, a_2, ... a_s, ... lar bilan belgilaymiz. Ushbu cheksiz ketma-ketliklarning barcha simvollarini standart { a_0, 1, q, U,CH} alfavit so'zlari bilan ifodalamiz. Bunda quyidagi qoidalar qabul qilinadi:

q_0 q so'zi bilan kodlansin;
q_1 qq so'zi bilan kodlansin;
q_2 qqq so'zi bilan kodlansin;

...

q_i qq...q (q lar i+1 ta) so'zi bilan kodlansin;

va h.k.

a_1 1 so'zi bilan kodlansin;
a_2 11 so'zi bilan kodlansin;

...

a_i 11...1 (1 lar i+1 ta) so'zi bilan kodlansin;

va h.k.

Standart alfavitda Tyuring mashinasi dasturini quyidagi qoidaga asosan SO'Z ko'rinishida ifodalash mumkin. Oldin dasturning barcha buyruqlari o'chiriladi. Buning uchun q_i $a_i \rightarrow q_j$ $a_m x$, $x \in$ {e,Ch,O'} yozuvlarda «→»

belgisi tushirib qoldiriladi. q_1, a_1, a_1, a_m harflar standart alfavitning mos harflariga almashtiriladi. Bundan keyin buyruq-so'zlar ketma-ketligi bitta so'z shaklida yoziladi.

Shunday qilib, har bir Tyuring mashinasini qandaydir chekli standart alfavitdagi chekli so'z bilan ifoda etish mumkin bo'lar ekan.

Chekli alfavitdagi barcha chekli so'zlar to'plami sanoqli bo'lgani uchun, Tyuring mashinalari soni ham shunga mos bo'ladi.

Endi barcha Tyuring mashinalarini nomerlaymiz. Buning uchun turli xil Tyuring mashinalari dasturlarini ifoda etuvchi standart alfavitdagi barcha so'zlarni fiksirlangan sanoqli cheksiz ketma-ketlik ko'rinishida joylashtiramiz. Bunda quyidagi qoidaga rioya etiladi. Oldin barcha bir harfli so'zlar yozib olinadi:

$\alpha_0, \alpha_1, \ldots \alpha_k$ (bu ketma-ketlik chekli buladi).

So'ngra ikki harfli so'zlar terib olinadi: $\alpha_{k+1}, \alpha_{k+2}, \ldots, \alpha_l$ (bunday so'zlar ketma-ketligi ham chekli bo'ladi), keyin uch harfli so'zlar keladi va h.k. Natijada barcha Tyuring mashinalari dasturlari ketma-ketligiga ega bo'lamiz:

$$\alpha_0, \alpha_1, \ldots, \alpha_l.$$

l sonini Tyuring mashinasi nomeri deb qabul qilamiz.

Endi A={1} alfavitda berilgan so'zlar to'plamidan qiymat qabul qiluvchi barcha funksiyalar to'plamini ko'rib chiqamiz. Boshqa tomondan, barcha mavjud bo'lishi mumkin bo'lgan Tyuring mashinalarini nomerlash yo'li bilan ushbu mashinalar to'plamini sanoqli ekanligini ko'rsatdik. Bundan Tyuring buyicha hisoblanuvchi funksiyalar to'plamining ham sanoqli ekanligi kelib chiqadi. Yuqorida ifodalangan funksiyalar to'plami esa sanoqlidir. Bundan Tyuring buyicha hisoblanuvchi funksiyalaming mavjudligi kelib chiqadi. Hech bir Tyuring mashinasida hisoblanmaydigan aniq funksiya ko'rsatsak, funksiyani hisoblovchi algoritm mavjud emasligini isbotlaydi.

A={1} alfavitdan olingan so'zlar uchun berilgan φ funksiyani quramiz. Ixtiyoriy k uzunlikdagi A={1} alfavitdan olingan α So'z uchun:

$$\varphi(\alpha) = \begin{cases} B_n 1, & \text{agar } A=\{1\} \text{ alfavitdagi n nomerli TM} \\ & \alpha \text{ so'zni } B_n \text{ so'zga aylantirsa,} \\ 1, & \text{aks holda.} \end{cases}$$

Teorema-1: *$\varphi(\alpha)$ funksiya Tyuring mashinasi buyicha xisoblanuvchi emas.*

Isbot: Aksini to'g'ri deb hisoblaylik. Ya'ni T Tyuring mashinasi mavjud bo'lib, uning standart alfaviti {a_0, 1, q, U, CH} bo'lsin va ushbu funksiyani hisoblasin. K - ushbu Tyuring mashinasining nomeri bo'lsin. α=11…1(1 lar soni k ta) bo'lganda $\varphi(\alpha)=\varphi(11\ldots.1)$ ga teng. Bunda so'z nimaga teng bo'lishini ko'ramiz. Faraz qilaylik T mashina 11…1 so'zni B_k so'zga almashtirsin. Bu B_k ham A={1} dan olingan. Bundan $\varphi(11\ldots1)=B_k$ ekanligi kelib chiqadi.

Ikkinchi tomondan, $\varphi(\alpha)$ funksiyaning ifodasidan $\varphi(1\ldots1)=B_k1$ ekanligi ma'lum. Bu kelib chiqqan ziddiyat $\varphi(\alpha)$ ni hisoblovchi Tyuring mashinasi mavjud emasligini isbotlaydi.

Algoritmik yechimsizlik muammosiga yana bir misol – o'z-o'ziga qo'llanuvchanlikni aniqlashdir.

Faraz qilaylik Tyuring mashinasi tasmasida uning o'z funksional sxemasi yozilgan bo'lsin. Agar mashina ushbu konfiguratsiyaga qo'llanuvchan bo'lsa, uni o'z-o'ziga qo'llanuvchi deb ataymiz, aks holda esa qo'llanilmas bo'ladi.

Teorema-2: *Tyuring mashinalari o'z-o'ziga qo'llanuvchanligini aniqlash muammosi algoritmik yechimsizdir.*

Isbot: Aksini faraz qilaylik. Tyuring tezisiga asoslanib, Bunday algoritmni hal qiluvchi Tyuring mashinasi mavjud deb hisoblaymiz. T – shunday Tyuring mashinasi bo'lsin. Uning tasmasiga mos usulda kodlangan u yoki bu Tyuring mashinasining dasturi kiritiladi. Bunda agar mashina o'z-o'ziga qo'llanuvchan bo'lsa, kiritilgan so'z mashina tomonidan o'z-o'ziga qo'llanuvchanlik haqidagi savolga tasdiq javobini anglatuvchi S simvolga aylantiriladi. Mashina o'z-o'ziga qo'llanilmas bo'lsa, uning dasturini ifoda etuvchi KIRISH so'zi yuqoridagi savolga inkor ma'nosini anglatuvchi A simvolga aylantiriladi.

Endi T_1 Tyuring mashinasini ko'rib o'tsak, ushbu mashina o'z-o'ziga qo'llanilmas kodlarni A ga aylantirsa, o'z-o'ziga qo'llanuvchi kodlarga esa T_1 mashina qo'llanilmas bo'lsin. Bunday mashina T mashina yordamida quriladi, agar uning dasturi quyidagicha o'zgartirilsa, ya'ni S simvol paydo bo'lgandan keyin, mashina to'xtash o'rniga, uni cheksiz marta takrorlasin. Demak, T_1 mashina har qanday o'z-o'ziga qo'llanilmas kodga qo'llanuvchan (A simvol xosil kilinadi), ammo o'z-o'ziga qo'llanuvchan kodlarga qo'llanilmasdir. Bu esa ziddiyatga olib keladi, chunki bunday mashina o'z-o'ziga qo'llanuvchan ham, qo'llanilmas ham bo'la olmaydi. Ushbu ziddiyat *Teoremani isbotlaydi*.

Ushbu isbotlangan teorema ba'zi boshqa umumiy muammolarning ham yechimsizligini isbotlaydi. Masalan, Tyuring mashinasi uchun qo'llanuvchanlikni aniqlash muammosi algoritmik yechimsizdir. U quyidagidan iborat: Qandaydir Tyuring mashinasi dasturi va konfiguratsiyasi berilgan. Ushbu konfigurasiyaga berilgan mashina qo'llanuvchanmi, yo'qmi, aniqlash kerak. Bu masalani yechish algoritmi mavjud bo'lganda, uning yordamida mashinaning o'z dasturi kodiga qo'llanuvchan ekanligini aniqlash mumkin bo'lar edi. Ammo yuqoridagi teoremaga asosan, bunday algoritm mavjud emas.

2.3.1 The divide-and-conquer approach

Many useful algorithms are ***recursive*** in structure: to solve a given problem, they call themselves recursively one or more times to deal with closely related subproblems. These algorithms typically follow a ***divide-and-conquer*** approach: they break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

The divide-and-conquer paradigm involves three steps at each level of the recursion:

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem.

The ***merge sort*** algorithm closely follows the divide-and-conquer paradigm. Intuitively, it operates as follows.

Divide: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.

Conquer: Sort the two subsequences recursively using merge sort.

Combine: Merge the two sorted subsequences to produce the sorted answer.

The recursion "bottoms out" when the sequence to be sorted has length 1, in which case there is no work to be done, since every sequence of length 1 is already in sorted order.

The key operation of the merge sort algorithm is the merging of two sorted sequences in the "combine" step. We merge by calling an auxiliary procedure MERGE(A, p, q, r), where A is an array and p, q, and r are indices into the array such that $p \le q < r$. The procedure assumes that the subarrays $A[p..q]$ and $A[q+1..r]$ are in sorted order. It ***merges*** them to form a single sorted subarray that replaces the current subarray $A[p..r]$.

Our MERGE procedure takes time $\Theta(n)$, where $n = r - p + 1$ is the total number of elements being merged, and it works as follows. Returning to our card-playing motif, suppose we have two piles of cards face up on a table. Each pile is sorted, with the smallest cards on top. We wish to merge the two piles into a single sorted output pile, which is to be face down on the table. Our basic step consists of choosing the smaller of the two cards on top of the face-up piles, removing it from its pile (which exposes a new top card), and placing this card face down onto

NAZORAT UCHUN SAVOLLAR

1. Algoritmik yechimsizlik nima?
2. O‘z-o‘ziga qo‘llanuvchanlik nima?
3. Qanday algoritmik yechimsiz muammolarni bilasiz?
4. Har bir usul bo‘yicha algoritm tuzishga misol ko‘rsating.
5. Algoritmni ishlab chiqish uchun yana qanday usullarni bilasiz?

the output pile. We repeat this step until one input pile is empty, at which time we just take the remaining input pile and place it face down onto the output pile. Computationally, each basic step takes constant time, since we are comparing just the two top cards. Since we perform at most n basic steps, merging takes $\Theta(n)$ time.

The following pseudocode implements the above idea, but with an additional twist that avoids having to check whether either pile is empty in each basic step. We place on the bottom of each pile a ***sentinel*** card, which contains a special value that we use to simplify our code. Here, we use ∞ as the sentinel value, so that whenever a card with ∞ is exposed, it cannot be the smaller card unless both piles have their sentinel cards exposed. But once that happens, all the nonsentinel cards have already been placed onto the output pile. Since we know in advance that exactly $r - p + 1$ cards will be placed onto the output pile, we can stop once we have performed that many basic steps.

```
MERGE(A, p, q, r)
 1  n_1 = q - p + 1
 2  n_2 = r - q
 3  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i - 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

In detail, the MERGE procedure works as follows. Line 1 computes the length n_1 of the subarray $A[p..q]$, and line 2 computes the length n_2 of the subarray $A[q+1..r]$. We create arrays L and R ("left" and "right"), of lengths $n_1 + 1$ and $n_2 + 1$, respectively, in line 3; the extra position in each array will hold the sentinel. The for loop of lines 4–5 copies the subarray $A[p..q]$ into $L[1..n_1]$, and the for loop of lines 6–7 copies the subarray $A[q+1..r]$ into $R[1..n_2]$. Lines 8–9 put the sentinels at the ends of the arrays L and R. Lines 10–17, illus-

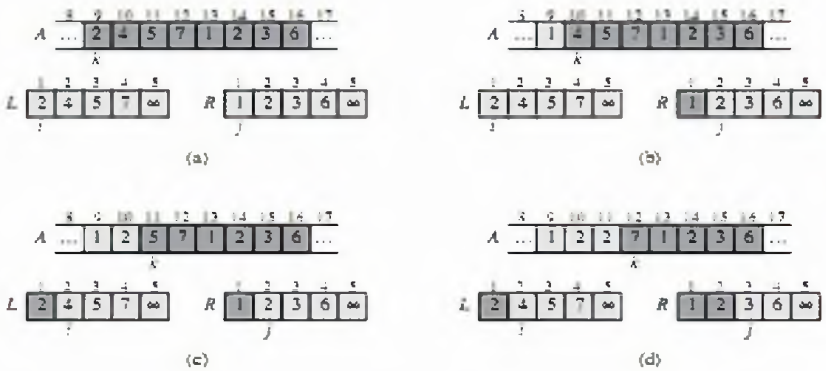


Figure 2.3 The operation of lines 10–17 in the call MERGE($A, 9, 12, 16$), when the subarray $A[9..16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$. After copying and inserting sentinels, the array L contains $\langle 2, 4, 5, 7, \infty \rangle$, and the array R contains $\langle 1, 2, 3, 6, \infty \rangle$. Lightly shaded positions in A contain their final values, and lightly shaded positions in L and R contain values that have yet to be copied back into A. Taken together, the lightly shaded positions always comprise the values originally in $A[9..16]$, along with the two sentinels. Heavily shaded positions in A contain values that will be copied over, and heavily shaded positions in L and R contain values that have already been copied back into A. (a)–(h) The arrays A, L, and R, and their respective indices k, i, and j prior to each iteration of the loop of lines 12–17.

trated in Figure 2.3, perform the $r - p + 1$ basic steps by maintaining the following loop invariant:

> At the start of each iteration of the **for** loop of lines 12–17, the subarray $A[p..k-1]$ contains the $k - p$ smallest elements of $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$, in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

We must show that this loop invariant holds prior to the first iteration of the **for** loop of lines 12–17, that each iteration of the loop maintains the invariant, and that the invariant provides a useful property to show correctness when the loop terminates.

Initialization: Prior to the first iteration of the loop, we have $k = p$, so that the subarray $A[p..k-1]$ is empty. This empty subarray contains the $k - p = 0$ smallest elements of L and R, and since $i = j = 1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

[3]

FOYDALANILGAN ADABIYOTLAR RO‘YHATI:

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. Сер: Классические учебники: COMPUTER SCIENCE. М.: МЦНМО, – 960с., 2004.
2. Вирт Н. Алгоритмы и структуры данных. С примерами на Паскале. Санкт-Петербург, 352с., 2005.

[3] **Thomas H. Cormen. Intruduction to algorithms. Massachusetts Institute of Technology. London 2009. (29-31 pp)**

2 MODUL. PASKAL DASTURLASH TILINING IMKONIYATLARI.
MA'RUZA № 5

MAVZU: PASKAL TILI DASTURLASH TILINING ALIFBOSI, BUYRUQLAR TIZIMI VA OPERATORLARI.

REJA:

1. Dasturlash tillari va ularning klassifikasiyasi, mashinaga mo'ljallangan va proseduraga mo'ljallangan dasturlash tillari, yuqori darajali dasturlash tillari.
2. Turbo Paskal va Paskal ABC dasturlash muhitini taqqoslash.
3. Paskal dasturlash tili strukturasi va alifbosi.
4. Paskal tilida o'zgaruvchilarni tavsiflash.
5. Paskal tilining standart funksiyalari va kalit so'zlari
6. Qo'zg'aluvchan va qo'zg'almas nuqtali o'zgarmaslar.
7. Foydalanuvchining kutubxona modullari.

Tayanch so'z va iboralar: Pascal ABC dasturlash tili, o'zgaruvchi, standart funksiyalar, o'zgarmaslar, biblioteka modullari.

1. Dasturlash tillari va ularning klassifikasiyasi, mashinaga mo'ljallangan va proseduraga mo'ljallangan dasturlash tillari, yuqori darajali dasturlash tillari

Dasturlash tillari paydo bo'lishidan oldin dasturlar *mashina kodlari*da (mashina kodi - protsessor tomonidan bajariladigan xotiradagi instruksiyalar ketma-ketligidir) tuzilar edi. Katta-katta dasturlarni tuzishda juda ko'p vaqt talab qilinar, ularni xatolarini tuzatish juda qiyin, modifikatsiyalash esa ko'p hollarda ilojsiz edi. Shularni hisobga olib, inson uchun tushinarli bo'lgan dasturlash tillarini tashkil qilish muammosi paydo bo'ldi. Dasturlash tillari inson uchun tushinarli bo'lgan dasturlarni tuzish imkonini beradi. Bunday dasturlarning matnini mashinalarlarda bajarish uchun, ularni mashina kodiga aylantirish zarur. Buning uchun *traslyator* deb ataluvchi maxsus dasturlardan foydalaniladi.

Traslyatorlar ikki xil ko'rinishda bo'ladi: *interpretator* va *kompilyator*. Interpretator - dasturning har bir operatorini oraliq kodga tarjima qilib, mashina kodiga aylantiradi va uni bajarishga kirishadi. Kompilyator - dastur matnini to'laligicha mashina kodiga aylantirib, uni bajarishga kirishadi.

Birinchi *dasturlash tili* - assembler bo'lib, bu til past pogonadagi til turiga kiradi. Dasturning har bir qatori-bitta mashina komandasiga mos tushadi. Assembler tilida katta-katta dasturlarni yozish juda qiyin, shuning uchun keyinchalik yuqori pogonadagi dasturlar yaratildi. Bularga Beysik, Pascal, Fortran, Ci va hakozolar kiradi. Pascal - 1969 yilda Syurix texnika universiteti professori N.Virt tomonidan yozilgan bo'lib, talabalarga dasturlar tuzishni o'rgatishga mo'ljallangan.

Satr holati aniq harakatlarni qanday tugmachalar yoki qatorlar yordamida bajarish mumkin ekanligini bildiradi. Ekrandagi kalit qatori

sichqoncha yordamida boshqariladi. Berilgan komandani bajarish uchun sichqoncha shu satrga o'rnatilib, chap tugmacha bosiladi. Masalan, asosiy menyuga o'tish uchun F10 yoki "F10 Menu" qatoriga sichqoncha ko'rsatgichi o'rnatilib, tugmacha bosiladi.

Turbo-Pascal tahrirlagichi bir nechta fayllarni bir vaqtning o'zida tahrirlash imkoniyatini beradi. Tahrirlagichga kirish uchun "File" menyusida "New" komandasini tanlash kifoya qiladi. Ekranda NONAME00.PAS nomli oyna paydo bo'ladi. Kursor oynani chap tepa burchagida o'rnatiladi. Bir vaqtning o'zida bir nechta tahrirlagich oynasini ochish uchun "New" komandasi bir necha marotaba bajariladi.

Tahrirlagich komandalari WD matn tahrirlagichi komandalari kabidir.

1-Jadval Kursorni siljitish komandalari

Tugmacha	Harakat
←,→	Bitta belgi chapga yoki o'ngga
↓, ↑	Bitta satr tepaga yoki pastga
Home	Satr boshiga
End	Satr oxiriga
PgUp, PgDn	Bitta bet tepada yoki pastga
Ctrl- ←	Bitta so'z chapga
Ctrl- →	Bitta so'z o'ngga
Ctrl-Home	Oyna boshiga
Ctrl-End	Oyna oxiriga
Ctrl-PgUp	Fayl boshiga
Ctrl-PgDn	Fayl oxiriga
Ctrl-Q-B	Blok boshiga

Tahrirlagichga matnni olib kirish ikki xil rejimlarda bajariladi: yozuvga qo'shish (insert) va qayta yozish (overwrite). Rejimlar almashinishi INS tugmacha yordamida amalga oshiralidi. Rejim almashinuvi ro'y bergan paytda kursor holati o'zgaradi. Qolgan komandalar guruhi tayyor matnlarni korrektirovka qilish uchun mo'ljallangan. Komandalar ro'yxati 2- jadvalda berilgan.

2- jadval. Ko'shish va o'chirish komandalari

Tugmacha	Harakat
Ins	O'chirish/yoqish qo'shish rejimi
Del	Kursor holatida belgi o'chirish
Backspace	Kursordan chapdagi belgini o'chirish
Ctrl-Y	Qatorni o'chirish
Ctrl-Q-Y	Kursor turgan joydan qator oxirigacha bo'lgan belgilarni o'chirish

Ctrl-N	Qatorni qo'shish

Bloklar bilan ishlash.

Blok bilan ishlashdan oldin matnda blokni ajratib olish kerak. Bu quyidagicha amalga oshiriladi: Kursorni ajratish kerak bo'lgan matni boshiga quyib CTRL-K-B ni birgalikda bosiladi, so'ngra kursorni blok oxiriga o'rnatib CTRL-K-K bosiladi. Keyin ekranda ajratilgan blok kerakli rangda paydo bo'ladi. Bloklarni ajratish uchun SHIFT-↑,↓,←,→ yo'naltiruvchi tugmachalardan yoki bo'lmasa sichqonchadan foydalanish mumkin. Bloklar bilan ishlash 3- jadvalda berilgan.

3- jadval. Bloklar bilan ishlash komandalari

Tugmacha	Harakat
Ctrl-K-C, Shift-Ins	Blokdan nusxa olish
Ctrl-K-Y, Ctrl-Del	Blokni o'chirish
Ctrl-K-V, Shift-Del	Blokni surib quyish
Ctrl-K-H	Blokni rangini ajratish
Ctrl-K-P	Blokni chop etish
Ctrl-K-R	Blokni diskdan o'qish
Ctrl-K-W	Blokni diskga yozish

Bloklarni tahrirlashda *Clipboard* nomli qo'shimcha komanda mavjud.

Bu komanda tahrirlash rejimida bir nechta oynalar bilan ishlash imkoniyatini beradi.

Masalan, 1- oynadan 2- oynaga blok matnidan nusxa ko'chirish uchun quyidagi harakatlar ketma-ketligi bajariladi: 1- oynada blokni ajratib, CTRL-INS bosiladi, so'ngra ALT-2 yordamida 2- oynaga o'tib, blokdan nusxa olish uchun SHIFT-INS bosiladi.

Bundan tashqari bloklarni tahrirlashda bosh menyudagi "Edit" maxsus komandasidan foydalanish mumkin. Bu quyidagicha bajariladi: "Edit" komandasi tanlangandan so'ng ekranda, misolda ko'rsatilganlarni bajarish uchun maxsus menyu paydo bo'ladi.

"Cut" qirqib olib tashlash komandasi o'zining harakatlari bo'yicha "Copy" komandasi kabidir, lekin farqi shundaki Clipboard ga blokdan nusxa olinganda blok joriy oynadan o'chib ketadi. Blokni joriy oynada asl holatiga takrorlash uchun SHIFT-INS bosiladi. Standart qadamlar ketma-ketligi: kiritish, saqlash, bajarish, tahrirlash - har qanday dastur tuzishni umumiy ssenariysini tashkil qiladi.

Tashkil qilish:

Kompilyator ishga tushgandan so'ng ekranda tahrirlagichni standart oynasi paydo bo'lmasa, yangi fayl tashkil qilish uchun "File" menyusidagi

"New" komandasi yoki ALT-F-N bajarilishi kerak. So'ngra tanish komandalar orqali joriy matnni dasturga kiritish lozim.

Saqlash:

Joriy matnni saqlash uchun "File" menyusidagi "Save" komandasidan foydalaniladi yoki F2 bosiladi. Ekranda fayl nomini so'rovchi oyna hosil bo'ladi. Nom kiritilgandan so'ng komandani bajarish uchun ENTER bosiladi va joriy katalogda shu nomli yangi fayl paydo bo'ladi.

Kompilyatsiya:

Dasturni saqlab quyilgandan so'ng kompilyatsiyalashga o'tish mumkin. Kompilyatsiyalashda matndagi sintaksis xatolar va boshqa xatolar tekshiraladi. Buning uchun ALT-F9 bosiladi yoki asosiy menyuning ("Compile") opsiyasiga o'tib, "Compile" komandasi bajariladi. So'ngra kompilyatsiyani bajarilishi haqida qo'shimcha ma'lumotni o'z ichiga olgan ("Compiling") oynasi paydo bo'ladi.

Agarda dastur matnida xatolik topilsa, u holda birinchi qator oynasida xatolik va xatolikni kelib chiqish sabablari haqida ma'lumot paydo bo'ladi. Bu holatda kursor avtomatik ravishda xatolik bor qatorga yoki undan keyingi qatorga o'tib qoladi.

So'nrga dastur matnidagi xatoliklarni tuzatib, yana bir marta " Compile" komandasini bajarish lozim. Agarda boshqa xatolik topilmasa, u holda kompilyator bajariluvchi (executable) fayl obrazini yaratadi. "Compile" meyusining "Destination" komandasi bajariluvchi faylni qayerda saqlashni: xotirada (memory) yoki diskda (disk) EXE fayli ko'rinishida saqlashni aniqlashga yordam beradi. O'zgartirilgan dastur matnini F2 bilan saqlab, so'ngra dasturni bajarishga o'tiladi.

Bajarish:

Dasturni bajarish uchun CTRL-F9 bosiladi yoki asosiy menyuning ("Run") opsiyasiga o'tib, ("Run") komandasi chaqiriladi. "Run" komandasi bajarilishidan oldin avtomatik ravishda dastur kompilyatsiyalanganmi yoki yo'qligi tekshiriladi, agarda kompilyatsiyalanmagan bo'lsa kompilyatsiyalanib, so'nga dastur bajariladi.

Dastur ishlash paytida IDE tasviri, kompilyator ishga tushmasdan oldingi DOS ekraniga o'rnini bo'shatib beradi. Dastur ishi tugagandan so'ng boshqarish yana IDE ga uzatiladi.

Natijalarni ko'rib chiqish:

Dastur natijalarini ko'rib chiqish uchun "Window" menyusidagi ("Output") komandasidan foydalaniladi. ("Output")ga kursorni olib kelib ENTER ni bosamiz, ekranda mos tartib raqamli oyna paydo bo'ladi. Ekranni to'la ko'rib chiqish uchun "Output" oynasini ekranga to'la yoyib chiqishni o'zi yetarli bo'ladi. Xuddi shu imkoniyatni qo'shimcha "Window" menyusining ("User screen") komandasi ham bajarishi mumkin.

Dasturga qayta murojat qilish:

Diskga yozilgan biror bir dasturni ochish uchun "File" (F3) menyusidagi ("Open") komandasidan foydalaniladi. Komanda bajarilganda joriy

katalogdagi barcha fayllar ro'yxati ekranga chiqadi va oddiy klaviatura yoki sichqoncha yordamida kerakli faylni tanlab olish mumkin. Qo'shimcha ravishda ish tugagandan so'ng, IDE da avtomatlashtirilgan holatda diskda barcha tahrirlanuvchi fayllar ro'yhatini saqlab quyish imkoniyati mavjud. Agada shu narsa sodir bo'lgan taqdirda keyingi ishdan oldin sizning oxirgi dasturingiz avtomatik tarzda xotiraga yozilib qoladi.

Integrallashgan muhit bilan tanishuv

Integrallashgan muhit, ko'poynali matnli redaktor, bosh menyu, sistemali menyu, fayllar bilan ishlash, ko'poynali redaktorlash, matnda izlash va almashshtirish, dasturlarni bajarish, dasturni kompilyatsiyalash, dastur xatolarini tuzatish, rejimlarni o'rnatish, qatorlar holati, "tezkor" tugmachalar, redaktorlash komandalari, Clipboard.

O'zidan oldigi komilyatorlardan farqlirok Turbo Pascal kopilyatori intelektual (the integrated development environment - IDE) *integrallashgan muhitni* o'z ichiga oladi. Kompilyatorning asosiy xususiyatlaridan biri, ko'p oynali sistema, *ko'p oynali matnli redaktor*, dialoglar oynasi (dialogs) va turli menyular va xokozolardir.

Kompilyator ishga tushgandan so'ng ekranda asosiy oyna paydo bo'ladi. Oynaning yuqori qatori gorizontal yo'laklar (the menu bar) ko'rinishidagi menyuga ajratiladi. Bunga Turbo Pascal kompilyatorining *bosh menyu*si deyiladi. Uning yordamida quyidagi komandalarga osongina murojat etish mumkin: *sistemali menyu(YO), fayllar bilan ishlash* ("File"), *ko'p oynali redaktorlash* ("Edit"), *matnda izlash va almashtirish* ("Search") , *dasturlarni bajarish* ("Run") , *dasturni kompilyatsiyalash* ("Compile"), *dastur xatolarini to'g'rilash* ("Debug"), *rejimlarni o'rnatish* ("Options"), oynalar bilan manipulyatsiyalar ("Window") va yordam ("Help"). Oynaning pastki qatori (the status line) *holatlar qatoriga* ajratilgan bo'lib, u yerda funksional tugmachalarning vazifalari ko'rsatilgan: F1 - Help, F2 - Save, F3 - Open, ALT-F9 - Compile, F9 - Make, F10 - Menu. Oynaning qolgan qismi maxsus belgilar bilan to'ldirilgan va IDE (the desktop) ishchi oblastini tashkil etadi.

Asosiy menyuning komandalariga uchta usuldan birortasi yordamida murojat etish mumkin: F10 ni bosib, menyu ichiga kirib, kerakli komandani tanlash mumkin. Tanlangan komanda ekranda mos rangda paydo bo'ladi. Uni bajarish uchun esa ENTER ni bosish kerak. Bundan tashqari menyuni boshqarish uchun sichqonchadan foydalanish mumkin. Va oxirida, asosiy menyudan biror-bir komandani tanlash uchun *"tezkor" tugmacha* (hotkeys) lardan foydalanish mumkin.

Har bir komandaning kalit so'zlariga qoida bo'yicha, bosh, bitta liter ajratilgan. Ajratilgan literlarning birortasi bilan ALT ni birgalikda bosib, mos komandani bajarishga kirishish mumkin. Masalan, "Compile" komandasini tanlash uchun ALT-C ni bosish kifoya, sistemali menyuga o'tish uchun esa, ALT-SPACEBAR bosiladi. Siz o'zingiz uchun qulay bo'lgan variantni tanlab olishingiz mumkin.

2. Turbo Paskal va Paskal ABC dasturlash muhitini taqqoslash

Paskal dasturlash tili o'zining soddaligi, mantiqiyligi va samaraligi tufayli bu til butun dunyoga tez tarqaldi. Hozirgi paytda barcha hisoblash mashinalari, hususan mikroEHMlar ham shu tilda ishlash imkoniga ega. Dasturlar matnining to'g'riligini osonlik bilan tekshirish mumkinligini, ularning ma'nosi yaqqol ko'zga tashlanishi va oddiyligi bilan ajralib turadi.

Paskal tili ancha murakkab va ko'p vaqt oladigan hisoblash ishlarini bajarishga mo'ljallangan tarkiblashtirilgan dasturlar tuzishga imkon beradi. Yana bir afzalligi shundan iboratki foydalanuvchi xattolikka yo'l quymasligi uchun yoki xatto yozib quygan bo'lsa, tez tuzatib olishi uchun dasturda ishlatilgan o'garuvchilar oldindan qaysi turga mansub ekanligi belgilab qo'yilgan bo'ladi. Shu bilan birga dasturning barcha elementlari haqida ma'lumot tavsiflash bo'limida mujassamlashgan bo'ladi. Operatorlar soni esa, minimal darajada kamaytirilgandir.

Algoratmik tillarning mashina tillaridan asosiy farqlari sifatida quyidagilarni ko'rsatishmumkin:

-mashina tili alifbosidan algoritmik til alifbosining o'ta kengligi;
-tuzilgan dastur matinning ko'rinish sifatini keskin oshiradi;
-ishlatilishi mumkun bo'lgan amallar majmui mashina amallari majmuiga bog'liq emas;
-bajariladigan amallar odam uchun qulay ko'rinishda, ya`ni amalda qabul qilingan matematik belgilashlarda beriladi;
-amallar operandlari uchun dasturchi tomonidan beriladigan shaxsiy ismlar qo'yish mumkinligi;
-mashina uchun ko'zda tutilgan malumot tiplaridan tashqari yangi tiplar kiritish imkoniyati yaratilganligi.

Shunday qilib, ma'lum ma'noda aytish mumkinki, algoritmik tillar mashina tiliga bog'liq emas.

Yo'qorida aytilganlardan kelib chiqqan holda ma'lum bo'ldiki algoritmik tilda yozilgan masala yechimining algoritmi to'g'ridan to'g'ri EHM da bajarilishi mumkin emas ekan. Buning uchun esa algoritm oldindan ishlayotgan EHM ning mashina tiliga translyator (kompiliyator yoki interpretator) yordamida o'girilishi lozim.

Til alifbosi shu tilgagina tegishli bo'lgan chekli sondagi belbilardan tashkil topgan. Dastur matnini yozishda faqat shu belgilardan foydalanish mumkin, boshqa belgilarni esa til tanimaydi yani ularda foydalanish mumkin emas.

Til sintaksisi alfovit harflaridan tashkil topgan bo'lib, mumkin bo'lgan konstruksiyalarni aniqlovchi qoidalar tizimidir. Mazkur tilda ifoda etilgan to'la algoritm va uning alohida hadlari shu konstruksiyalar orqali ifoda qilinadi. Shunday qilib, belgilarning har qanday ketma-ketligini, hamda mazkur tilning matni to'g'riligi yoki noto'g'riligi til sintaksisi orqali bilib olamiz.

Til semantikasi algoritmik tilning ayrim konstruksiyalari uchun qoidalar tizimini tushuntirishga xizmat qiladi.

1981 – yilda Paskal tilining xalqaro standarti taklif etildi. Paskal tili Borland firmasi tomonidan yaratilagan.

Integrallashgan muhit - dasturlashga yordamlashuvchi dastur bo'lib, quyidagi vazifalarni o'z ichiga oladi:

- vazifalari dastur matnini kiritish imkonini beradi;
- kiritilayotgan dastur matnini tashqi xotirada saqlb turadi;
- uni ishga tushirishi uchun translyatori bor;
- sintaktik xatolarni momentalna aniqlaydi.

Mazkur katalogda TURBO.EXE fayliga murojaat qilingandan so'ng ekranda Turbo-Paskal muhitining o'z menyu satriga ega bo'lgan tahrir qilish sahifasi ochiladi.

Turbo – Paskal integrallashgan muhiti interfeysida *menyular satri, ishchi maydoni, ma'lumot satri* mavjud.

1 – rasm.

1 – rasmda Paskal dasturining oynasi ko'rinib turibdi.

Menyu satrida alohida vazifalariga ega bo'lgan 9 ta bo'limlar mavjud.

File, Edit, Search, Run, Comple, Debug , Options, Windows, Help.

2 – rasm.

Har bo'lim o'z bandlariga ega bo'lib, ularning ichida ... belgi bilan tugaganlari alohida muloqot darchalariga ega bo'ladilar.

File bo'limiga murojaat etilganda hosil bo'lgan majmuada Open... F3(2 - rasm) bandi kompyuter xotirasidan Paskal fayllarini ekranga chaqirish uchun

mo‘ljallangan. Mazkur band faollashtirilganda muloqot darchasi hosil bo‘lib, u yerda kerakli fayl katalog ichidan axtariladi.

New yordamida yangi dastur matnini kiritish uchun oyna ochiladi.

Save F2 dasturni xotiraga kiritadi.

Save as… dasturni biror nom ostida xotiraga kiritadi.

Save all barcha fayllarni xotiraga kiritadi.

Change dir… yangi katalog hosil qiladi.

Print dastur matnini chop etadi.

Printer setup… dastur hisoblashi davomida kopyuter imkoniyatlaridan foydalanish darajasi haqida ma’lumot beradi.

Dos shell dasturdan vaqtinchalik operatsion tizimga chiqib turish imkoniyatini yaratadi.

Exit (Alt + x) NC ga chiqiladi.

Edit Paskal dasturlarini tahrir qilish vazifasini bajaradi. Tahrir qilish davrida belgilangan bo‘laklar ustida amal bajarish uchun klaviaturadagi tugmalarning quyidagi majmuasidan foydalanishimiz mumkin:

Ctrl+K+B -ajratiluvchi bo‘lakning boshini belgilash;

Ctrl+K+K - ajratiluvchi bo‘lakning oxirini belgilash;

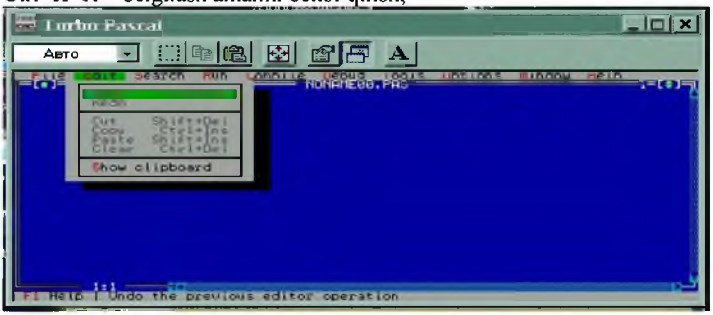
Ctrl+K+C – belgilangan bo‘lakning nusxasini olish;

Ctrl+K+V – belgilangan bo‘lakni boshqa joyga ko‘chirish;

Ctrl+K+Y – belgilangan bo‘lakni o‘chirish;

Ctrl+K+P – belgilangan bo‘lakni chop etish;

Ctrl+K+H – belgilash amalini bekor qilish;



3 – rasm.

Undo - belgilangan bo‘lakni buferda saqlanishi;

Cut (Shift +del)belgilangan bo‘lakni olib tashlash;

Copy - xotiraga bo‘lakning nusxasini o‘tkazish;

Paste - bo‘lak nusxasini dasturda hosil qilish;

Show clipboard - almashish buferi mazmunini ko‘rish;

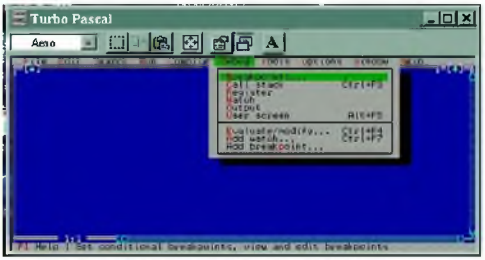
Clean - sahifani tozalash;

4 – rasm.

Search bo‘limi belgi va so‘zlarni axtarish va almashtirish vazifalarini bajaradi:
Find - dasturda belgi va so‘zni axtarish;
Replace - topilgan belgini o‘zgartirish;
Search again - amalni yangidan bajarish;
Goto line number - raqami ko‘rsatilgan qatorga o‘tish;
Find protsedure - kichik dasturni axtarish;
Find error - hisoblash xatoliklarini aniqlash.

Mazkur bo‘limning bandlariga murojaat qilinganda muloqot darchasi hosil bo‘lib, u yerda bajarilayotgan vazifalarni ko‘lami belgilanadi, qaralayotgan soha chegaralanadi.

Run bo‘limida tahrir qilingan dasturni hisobga o‘tkazish bandlari jamlangan:
Run - dasturni hisobga o‘tkazish;
Progrem reset – tahrir qilishni to‘xtatish;
Goto cursor - kursor turgan joygacha hisoblash;
Trace into - hisoblash algoritmini ko‘rish;
Step over - satrlab hisoblash;
Parametrs - dastur parametrlarini aniqlash.
Trace unto bandi dasturni belgilangan algoritm bo‘yicha qadamlab hisoblaydi, natijada mavjud kamchiliklarni aniqlash osonlashadi.
Step over bandi yuqoridagi bandga o‘xshash vazifani amalga oshirsa-da, hisoblash davomida



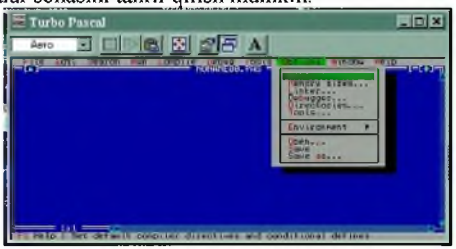
Debug bo‘limida 4 ta band bo‘lib, ularning har biri dastur hisoblashida yuzaga keluvchi xatoliklarni aniqlashni osonlashtirish vazifasini bajaradi.

Evaluate/modify-o‘zgaruvchi qiymatlarini baholash;
Watches - to‘xtash joyi va qiymatni ko‘rish;
Toggle breakpoint - to‘xtash satrini tanlash;
Breakpoints - to‘xtash nuqtasi amallari.

Evaluate/modify … bandi dastur hisoblashida oraliq o‘zgaruvchilar qabul qilgan qiymatlarni ko‘rish uchun mo‘ljallangan bo‘lib, murojaat etilganda ekranda muloqot darchasi hosil bo‘ladi. Mazkur darchaning birinchi satrida qaralayotgan o‘zgaruvchi yoziladi va keyingi qatorda uning joriy qiymati hosil bo‘ladi.

Watches bandi muloqotli darchasi quyidagi ko‘rinishda bo‘ladi:

Mazkur bandning satrlaridan foydalanib satrlab hisoblash usulida kerakli o‘zgaruvchining qabul qilayotgan qiymatlari uzluksiz kuzatib turiladi. Zarur bo‘lganda ifodalar sohasini tahrir qilish mumkin.



Options bo‘lim Turbo-Paskal muhiti ayrim xossalarini boshqarish uchun mo‘ljallangan:
Compiler - kompilyator;
Memory sizes - xotira hajmi;
Linker - moslashtiruvchi (kompanovshik);
Directories - jadvallar;
Environment - faoliyat sharti;
Save options - opsiyalarni diskka yozish;
Retrieve options - opsiyalarni diskdan o‘qish.

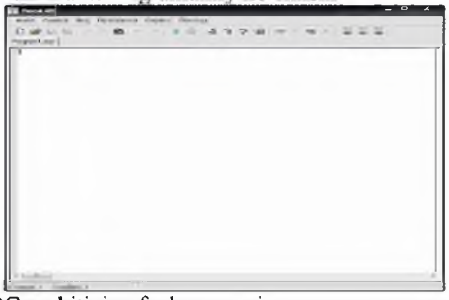
Compilier bandi muloqot darchasi yordamida hisoblash paytida qiymatlar o‘zgarishi oraliqlari, kiritish va chiqarish nazorati, matematik soprotsessorni qo‘shish va shu kabi vazifalarni amalgan oshirish mumkin. Bunda [ ] ichida bo`lishi kerak.

Turbo-Paskal menyusining navbatdagi bo‘limlari kopyuter ekranidan natija olishni maqbullashtirish va tizim haqida kerakli ma’lumotlarni tavsiya qilish vazifalarini bajaradi.

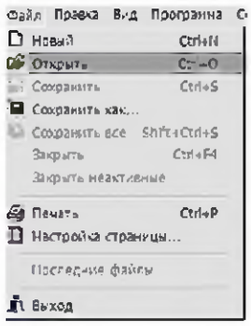
PaskalABC muhitiga kirish uchun quyidagi bosqichlarni amalga oshiramiz

Пуск-программы-PascalABC buyrug‘i tanlanadi

Paskal dasturlash tili muhitining umumiy ko‘rinishi

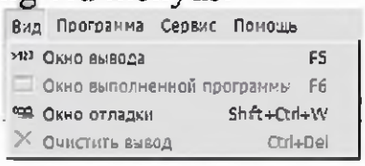


1. Paskal ABC muhitining fayl menyusi

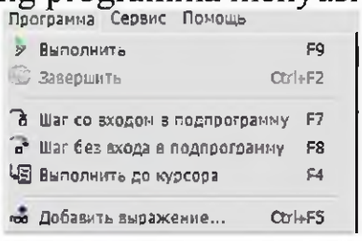


2. Paskal ABC muhitining pravka menyusi

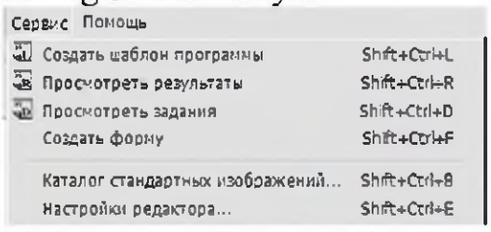
3. Paskal ABC muhitining vid menyusi



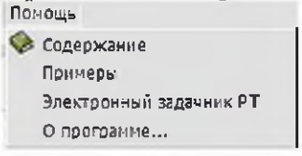
4. Paskal ABC muhitining programma menyusi



5. Paskal ABC muhitining servis menyusi



6. Paskal ABC muhitining pomosh menyusi



3. Paskal dasturlash tili strukturasi va alifbosi

Paskal tili alfaviti

Tanlab olingan algoritm asosida qo‘yilgan masalani kompyuterda yechish uchun qanday algoritmik tilning o‘z alifbosi, buyruqlar majmuasi va maxsus dasturlar jamlangan kutubxonasi bo‘lishi zarur. Paskal tili alifbosini uch qismga bo‘lish mumkin:

Paskal tili alifbosiga quyidagilar kiradi:

1. 26 ta lotin alifbosi harflari: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,S,T, U,V,W,X,Y,Z va rus alifbosi harflari.

2. Arab raqamlari: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. Nol soni O harfidan farq qilishi uchun dastur tuzishda uning ustiga chizib yoziladi.

3. Arifmetik amallar. Paskal tilida quyidagi arifmetik amal belgilari mavjud: ko'paytirish (*), masalan: A*B; bo'lish (/), masalan: A/B; qo'shish (+), masalan: A+B; ayirish (-), masalan: A-B. Paskal tilida darajaga ko'tarish amali yo'q. Shuning uchun ham sonlarni butun darajaga ko'tarish (daraja ko'rsatkichi katta son bo'lmasa) ularni bir necha marotaba ko'paytirish yo'li bilan amalgam oshirish mumkin. Haqiqiy darajaga ko'tarish (agar asos musbat son bo'lsa) logarifmlash yo'li bilan amalgam oshiriladi.

$$x^n=e^{n\ln x} \quad \text{yoki} \quad x^n=10^{n\lg x}$$

4. Munosabat amal belgilari:

Paskal belgisi	Matematik ko'rinishi	Ma'nosi
=	=	Teng
<>	≠	Teng emas
<	<	Kichik
<=	≤	Kichik yoki teng
>	>	Katta
>=	≥	Katta yoki teng

5. Maxsus belgilar: . (nuqta); , (vergul); ; (nuqtali vergul); : (ikki nuqta); oddiy, kvadrat va figurali qavslar: ( ), [ ], { }; probel yoki bo'sh joy tashlash, ' (apostrof); " (qo'shtirnoq) va hokazo.

6. Xizmatchi so'zlar: AND-va, ARRAY-massiv, BEGIN-boshlamoq, CASE-variant, CONST-o'zgarmas, DIV-butunga bo'lish, DO-bajarmoq, DOWNTO- gacha, ELSE-aks holda, END-tamom, FILE-fayl, FOR-uchun, FUNCTION-funksiya, GOTO-ga o'tish, IF-agar, IN-ga, LABEL-belgi, MOD-modul, NOT-yo'q, OF-dan, OR-yoki, PROGRAM-dastur, RECORD-yozuv, REPEAT-takrorlamoq, SET-to'plam, THEN-u holda, TO-gacha, TYPE-turi, UNTIL-gacha, VAR-o'zgaruvchi, WHILE-hozircha.

Foydalanuvchi tomonidan bajearilishi lozim bo'lgan ma'lum xarakatni elektron hisoblash mashinalariga maxsus so'zlardan tashkil topgan operatorlar `rdamida yetkazib amalga oshirish mumkin. Demak, kompyuter uchun operator bajarilishi so'zsiz shart bo'lgan buyruqdir.

Operatorlar algoritmik tillarda asosiy tushuncha bo'lib, o'z navbatida ikki guruxga bo'linadi: oddiy va murakkab operatorlar.

Oddiy operatorlar jumlasiga begin, end, Uses, const, label kabi operatorlar kiritilishi mumkin.

Murakkab operatorlar bir necha asosiy operatorlarni o'z ichiga oladi.

Kompyuterda biror masalani yechish uchun boshqa dasturlarga, tashqi qurilmalarga murojaat qilish mumkin, o'zgarmas yoki yangi o'zgaruvchilarning ko'rinishini e'lon qilish mumkin va xakozo.

Shunday qilib **Paskal tilidagi dastur strukturasi** quyidagi ko'rinishda bo'ladi

Program Programma mavzusini berish
Uses Ishlatilayotgan kutubxona bo'limlari (modullari)
Label Dasturning asosiy qismida ishlatilayotgan belgi (metka) larni e'lon qilish
Const O'zgarmaslarni e'lon qilish
Type Yangi o'zgaruvchilarning turini muomalaga kiritish
Var Asosiy dasturda muomalada bo'ladigan o'zgaruvchilarni e'lon qilish
Procedure, Function – Protsedura va Funksiyalarni e'lon qilish.
Begin
Dasturning asosiy qismi
End.

Demak, har qanday dastur yuqorida berilgan asosiy tuzilmaning xususiy xoli bo'lishi mumkin va ular o'z navbatida Paskal tiliga xos bo'lgan asosiy tushunchalar asosida xosil qilinadi.

Paskal tilida o'zgaruvchilarni tavsiflash

Ma'lumki, har qanday qiymat yoki belgi bilan ish ko'rish uchun eng avvalo ularga xotirada joy ajratish zurur bo'ladi. Buning uchun ishlatilishi zarur bo'lgan o'zgaruvchi yoki o'zgarmaslar Paskal tilida e'lon qilinishi kerak. Ko'pchilik xollarda dasturlarda o'zgarmas qiymatlar bilan ish ko'rishga to'g'ri keladi. Masalan, n=20, e=2.71 kabi sonlar Paskal tilida quyidagicha e'lon qilinadi.

Const pi=3.14; n=20; e=2.71;

Umumiy holda o'zgaruvchilar var (variable) operatori orqali qabul qilishi mumkin bo'lgan qiymatiga qarab turlarga bo'linadi. Butun sonlar ishlatilishi chegarasiga qarab har xil e'lon qilinishi mumkin.

Butun sonli tipda berilganlar arifmetik ifodalarda qo'llaniladigan qiymatlardan iborat bo'lib, 1 dan 4 baytgacha bo'lgan xotirani egallaydi.

Tipi	Diapazoni	Kerak bo'lgan xotira (bayt)
Byte	0..255	1
Shortint	-128..127	1
Integer	-32768..32767	2
Word	0..65535	2
Longint	-2147483648..2147483647	4

Haqiqiy tiplar matematik ifodalarda qo'llaniladigan haqiqiy qiymatlardan iborat bo'lib, 4 dan 6 baytgacha xotirani egallaydi.

Haqiqiy sonlar uchun qo'yilgan masalada yechimning aniqlik darajasiga qarab quyidagi operatorlar yordamida identifikatorlar e'lon qilinadi:

Identifikator turi	qiymatlar oralig'i	Aniqlik darajasi	Egallagan shajmi

Real	2.9e –39 .. 1.7e38	11 – 12	6 bayt
Single	1.5e –45 .. 3.4e38	7 – 8	4 bayt
Double	5.0e –324.. 1.7e308	15 – 16	8 bayt
Extended	3.4e –4932 ..1.1e4932	19 – 20	10 bayt
Comp	$-2e+63^{+1}..2e+63^{-1}$	10 – 20	8 bayt

Belgili tiplar

Literli (belgili) tip ShK ni kod jadvalining qiymatlarini aniqlaydi. Literli tip o'zgaruvchisi uchun 1 bayt kerak bo'ladi.

Misol.

```
VAR
  Ch: char;
```

Dasturda char tipidagi o'zgaruvchilar va konstantalar apostrof ichiga olib yoziladi. Misol uchun, 'A' A harfini beradi, ' ' - joy tashash, '.' - nuqta vergul.

Bulev tipi

Bulev tipi ikki xil qiymat bilan beriladi: True (rost) va False (yolg'on). Bu qiymatlar mantiqiy ifodalarda va munosabat ifodalarida keng qo'llaniladi.

Jadval. Bulev tipi

Tipi	Diapazoni	Kerak bo'lgan xotira
Boolean	True, False	1

Foydalanuvchining tiplari

Pascal tilida standart tiplardan tashqari foydalanuvchi tomonidan aniqlangan skalyar tiplar mavjud. Bularga sanab o'tiladigan va interval tiplar kiradi va xotirada 1 bayt joy egalaydi. Shuning uchun foydalanuvchining tiplari 256 belgidan oshmasligi kerak. Ularni qo'llash dastur ko'rinishini ancha o'zgartiradi, xatolarni topish osonlashadi va xotira tejaladi.

Sanab chiqiladigan tiplar shu tipdagi berilganlar qanday qiymatlarni qabul qilsa, shu qiymatlarni sanab chiqish orqali beriladi. Alohida qiymatlar vergul orqali ajratiladi, hamma ro'yxat esa qavs ichiga olib ko'rsatiladi.

Yozilishi: TYPE

```
    <tip nomi> = (<1-qiymat, 2- qiymat,..., n  - qiymat>);
  VAR
    <identifikatori,...> : < tip nomi>;
```

Misol.

```
TYPE
  Gaz = (C, O, N, F);
  Metall = (Fe, Co, Na, Cu, Zn);
VAR   G1, G2, G3 : Gaz;
  Met1, Met2 : Metall;
  Season: (Winter, Spring, Summer, Autumn);
```

Bu misolda foydalanuvchi tipining ikkita Gaz va Metall ko'rinishidagi yozuvlari berilgan. Ularni qiymatini aniqlash - Mendeleyev D.I davriy sistemasidagi gaz va metallarning belgilanishini beradi. G1, G2, G3 va Met1, Met2 o'zgaruvchilari yuqorida keltirilgan qiymatlarning bittasini qabul qilishi mumkin. Boshqa qiymatlarni qabul qilish dasturni uzilishiga olib keladi. Uchinchi tipdagi sanab o'tiladiganlar ananim (nomsiz) va ular VAR da qiymatlarni sanab o'tish orqali beriladi.

Season shu tipdagi o'zgaruvchi bo'lib, Winter, Spring, Summer va Autumn qiymatlarini qabul qilishi mumkin.

Bir xil tipdagi sanab o'tiladigan qiymatlar uchun munosabat va mantiqiy operatsiyalarni qo'llash mumkin. Tartiblash yozuv tipi elementinining tartib nomeri orqali amalga oshiriladi. Masalan, Winter < Spring ifodasi rost bo'ladi chunki, yozuv tipida Spring Winter ga nisbatan katta tartib nomeriga ega.

Pascal boshqa tiplarga nisbatan farqlirok, foydalanuvchining sanab o'tiladigan tiplarida kiritish-chiqarish operatsiyalarini qo'llamaydi. Kerak bo'lgan paytda foydalanuvchining o'zi kiritish-chiqarishni tashkil qiladi. Sanab o'tilgan tiplar bilan ishlash uchun Pascal tilida Succ, Pred, Ord standart quyi dasturlaridan foydalaniladi.

Interval tip, berilgan o'zgaruvchi uchun qiymatlar chegarasi diapozonini aniqlovchi ikkita konsantani berish imkonini beradi.

Kompilyator interval tipdagi o'zgaruvchilarda har bir operatsiyadan keyin o'rnatilgan ichki diapazonda o'zgaruvchining qiymati qoladimi yoki yo'qmi, tekshirish qism dasturini generatsiya qiladi. Ikkala konstanta ham standart tiplarning birortasiga (real dan tashqari) tegishli bo'lishi shart. Birinchi konstantaning qiymati albatta ikkinchi kostanta qiymatidan kichik bo'lishi shart.

Yozilishi: TYPE
<tipning nomi> = <1-konstanta> . . <2-konstanta>;
VAR
<identifikatori,...> : <tip nomi>;

Misol.

```
TYPE
  Days = 1 .. 31;
VAR
  RabDay, BolnDay : Days;
```

Bu misolda RabDay va BolnDay o'zgaruvchilari Days tipida bo'lib, ular 1...31 diapazonda har qanday qiymatlarni qabul qilishi mumkin. Diapazondan chiqish dastur o'zilishiga olib keladi. Interval tipni boshqacharoq, universal usul bilan ham aniqlash mumkin. Bu usulda diapazon chegarasini konstanta qiymatlari bilan emas, nomi orqali aniqlash mumkin:

```
CONST
```

```
 Min = 1; Max = 31;
TYPE
  Days = Min .. Max; VAR
RabDay, BolnDay : Days;
```

Paskal tilining standart funksiyalari va kalit so'zlari

Har bir algoritmik tilning dastur matnini yozish qoidalari turlicha bo'ladi. Dasturlash tillaridan eng soddasi Beysik tilining ma`lum versiyalarida dasturning bar bir operatori qat`iy aniqlangan qator raqamlari orqali yoziladi. Paskal tilida esa operatorlar ketma-ket yozilib, o'zaro «;» belgisi bilan ajratib boriladi. Bundan tashqari, yozilgan dasturning o'qishga oson va undan foydalanish qulay bo'lishi uchun dasturda «matnni ajratish» tushunchasi (bo'sh joy, qatorning tugashi va izohlar) dan foydalaniladi.Bo'sh joy (probel) grafik tasvirga ega bo'lmagan belgi bo'lib, qatordagi bo'sh joyni anglatadi. Lekin, bo'sh joy belgisi o'zining sonli kodiga ega va dastur matnidagi boshqa belgilar kabi komputerga kiritiladi.Qator oxiri (tugashi) boshqaruvchi belgi bo'lib, u ham grafik tasvirga ega emas. Ma`lumki, dastur matnini yozish davomida uni tabiiy ravishda yangi qatorlarga ajratilib yoziladi. Chunki, shu matn yozilmoqchi bo'lgan qog'ozning ham, komputer ekranining ham o'lchamlari cheklangan. Dastur matnini alohida qatorlarga ajratmay yozish ham mumkin, lekin bir satrga 256 tadan ortiq belgi sig'maydi. Dastur matnini alohida qatorlarga ajratish dastur tuzuvchining xohishiga qarab bajariladi. Ma`lum bir qator tugamay turib, yangi qatorga o'tish uchun «qator oxiri» tugmachasi bosiladi. Bu tugmacha ham o'zining maxsus sonli kodiga ega.

Izohlar dasturni o'qishga oson bo'lishi, uni qiynalmay tekshirib, yo'l qo'yilgan xatolarni to'g'rilash va dasturda bajarilayotgan ishlarni tushuntirib borish uchun qo'yiladi. Izohsiz yozilgan dasturni hujjat sifatida qabul qilinmaydi. Muvaffaqiyatli qo'yilgan izoh dasturning va dasturchining katta yutug'i hisoblanadi. Izohlar ixtiyoriy vaqtda dastur matniga kiritilishi yoki olib tashlanishi mumkin. Bu bilan dasturning ishi o'zgarib qolmaydi. Izohlarni «{» va «}» qavslari ichiga olinib yoziladi.Dastur «matn ajratgich»laridan foydalanishning quyidagi qoidalariga amal qilish lozim:tilning ketma-ket yozilgan ikkita konstruksiyasi orasiga albatta bo'sh joy yozilishi kerak; ajratgichlarni xizmatchi so'zlar, sonlar va ismlar orasiga qo'yish maqsadga muvofiq emas.

Paskal tilida standart funksiyalarning berilishi quyidagi jadvalda keltirilgan

Matematikada	*Paskalda*	*Izoh*
+, -, ·, ÷	+, -, *, /	qo'shish, ayirish, ko'paytirish, bo'lish
Sin x	Sin(x)	Sinus
Cos x	Cos(x)	Kosinus
tg x	Sin(x)/ Cos(x)	Tangens

Ctg x	Cos(x)/ Sin(x)	Kotangens
arctg x	arctan(x)	Tangens
|x|	Abs(x)	X ning absolyut qiymati
[x]	Int(x)	int(5.2)=5 butun qismi
lnx	Ln(x)	Natural logarifm
e^x	exp(x)	E= 2,71 exponent son
x ni kasr qismi	Frac(x)	Frac(5.2)=0.2 kasr qismi
x ni yaxlitlash	Round(x)	Round(3.245)=3.25
x ning butun qismi	trunc(x)	trunc(3.2)=3
$\frac{x}{y}$ dagi qoldiq	x mod y	10 mod 3=1
$\left[\frac{x}{y}\right]$	x div y	10 div 3=3
x^2	Sqr(x), x*x	x ni kvadrati
$\sqrt{x}$	Sqrt(x)	X ni kvadrat ildizi
x<y , x>y	x<y , x>y	x kichik y dan, x katta y dan
$x \geq y$	x >= y	X katta yoki teng y ga
x<y<z	x<y and y<z	x kichik y dan va y kichik z dan
$4 \cdot 10^9$	4e9	O'n darajasi

Darajaga ko'tarish amali bo'lmaganligi uchun x^y ni paskalda tasvirlash uchin quyidagi shakl almashtiramiz $x^y = e^{\ln x^y} = e^{y \ln x}$. Demak , $x^y = e^{y \ln x}$ Paskalda exp(y*ln(x)). Bu erda x>0. agar x<0 bo'lsa – exp(y*ln(x)) .

Shu formuladan foydalanib $\sqrt[y]{x}$ ni paskal ko'rinishida yozaylik. Buning uchun shakl almashtirishni amalgam oshiramiz:

$\sqrt[y]{x} = x^{\frac{1}{y}}$ ko'rinishiga ega. Demak, exp((1/y)*ln(x)) ko'rinishida yoziladi.

Simvollar uchun quyidagi funksiyalar ishlatiladi:

Chr(n) - n tartib nomeriga mos keluvchi belgini aniqlaydi
Ord(x) - x belgining tartib nomerini aniqlaydi
Pred(x) - x dan oldingi belgining tartib nomerini aniqlaydi
Succ(x) - x dan keyingi belgining tartib nomerini aniqlaydi
Keltirilgan funksiyalar Paskal tilida maxsus funksiyalar deb ataladi.

Paskal tilining kalit so'zlari

And-va
Array –massiv
Begin-boshlash
Case-variant
Const-o'zgarmas
Div –qoldiqsiz butun bo'lish
File –fayl
For –uchun
Function –funktsiya
Procedure –prosedura
Goto –ga o'tish
If –agar

Do –bajarish
Downto –gacha kamaytirish
Else –aks holda
End –tamom
In –ga tegishli
Label-belgi
Mod-butun qoldiqli bo'lishdagi qoldiq.
To –gacha ko'paytirish

So'zlar rezervlashgan so'zlarga, *standart indentifikatorlarga va foydalanivchining indentifikatorlariga* bo'linadi.

Rezervlashgan so'zlar tilning tashkiliy qismi hisoblanib, aniq ma'noga ega bo'ladi. Quyida Pascal versiyasidagi ShEHM lar uchun rezervlashgan so'zlar ro'yxati keltirilgan:

absolute	and	array	begin	case
const	end	div	do	external
file	for	forward	function	goto
inline	interface	interrupt	mod	nil
procedure	type	program	record	unit
until	label	repeat	uses	set
var	shl	while	not	shr
with	if	of	string	xor
downto	implementation	or	then	else
in	packed	to		

Standart identifikatorilar oldindan aniqlangan o'zgaruvchilarni, konstanta, protsedura va funksiyalarni belgilashga xizmat qiladi. Masalan, Sin(x) standart indentifikatori, berilgan burchak sinusini hisoblash funksiyasini chaqiradi. Har qanday standart identifikatorini rezervlashgan so'zlardan farqi shundaki, uni oldindan aniqlab olish mumkin. Lekin bu ko'p holatlarda xatolikga olib keladi. Shuning uchun amaliyotda standart identifikatorilardan ularni o'zgartirmasdan foydalangan maqulroqdir.

Foydalanivchining identifikatorilaridan dasturchi metka, konstanta, o'zgaruvchilar, protsedura va funksiyalarni belgilashda foydalanadi. To'g'ri tanlangan identifikatori dastur tushinishni, o'qishni osonlashtiradi va dasturni modifikatsiyalashda xato qilish extimolini kamaytiradi.

Masalan, oy, kun, yilni D harfi yoki boshqa biror bir belgidan ko'ra Data identifikatori bilan begilash qulayroqdir.

Identifikatorilardan foydalanishning umumiy qoidalari mavjud:

1. Identifikatori faqat harf yoki chiziqcha belgisi bilan boshlanadi (bundan son yoki harf bilan boshlandigan metka mustasno).
2. Identifikatori harf, son va chiziqcha belgisidan tashkil topishi mumkin (probel, nuqta va maxsus belgilardan foydalanish mumkin emas).
3. Ikkita identifikatori oraligida hech bo'lmaganda bitta probel bo'lishi shart.
4. Identifikatorilar uzunligi 127 ta belgidan iborat, lekin faqat oldingi 63 tasigina hisobga olinadi xolos.

1 graph - xato, identifikatori son bilan boshlandi.

Nomer. Doma - xato, identifikatorida nuqta ishtirok etayapti.

Konstantalar va o'zgaruvchilar

Konstanta, o'zgaruvchi, tiplashgan konstanta

Har qanday dastur qandaydir berilganlar bilan ishlagan taqdirdagina ma'noga ega bo'ladi. Xuddi boshqa dasturlash tillari kabi, Pascal tili ham konstanta yoki o'zgaruvchi ko'rinishidagi berilganlar bilan ish olib boradi. Shunday qilib, dasturdagi har bir element *o'zgaruvchi* yoki *konstanta* bo'ladi. Konstanta va o'zgaruvchilar uzlarining identifikatorilari (nomlari) orqali aniqlanadi va shu nomlar orqali ularga murojat etiladi.

Konstantalar deb dastur boshida e'lon qilingan va dastur oxirigacha o'zgarmaydigan qiymatlarga aytiladi. Konstantani aniqlash uchun rezervlashgan CONST so'zidan foydalanamiz.

Yozilishi: CONST

<identifikatori> = <konstantaning qiymati>;

Masalan.

```
CONST
  Max = 1000;
  Vxod = 'Segment 5';
```

O'zgaruvchilarning konsantantalardan farqi shundaki, ular o'z qiymatlarini dastur ishi davomida o'zgartirishi mumkin. Har qanday o'zgarivchilar va konstantalar aniq bir berilganlar tipiga kiradi. Konstantalar tipini avtomatik tarzda kompelyatorlar yordamida aniqlanadi. O'zgaruchilarning tipi ular bilan ishlashdan oldin e'lon qilinishi zarur. O'zgaruvchilarni e'lon qilish uchun VAR so'zi qo'llaniladi.

Yozilishi: VAR

<identifikatori> : <tip>;

Masalan.

```
  VAR
Sum1, Sum2: real;
```

O'zgaruvchining nomi "qobiq" hisoblanib, uni qiymatlar bilan to'ldirish mumkin, lekin konstantalar bilan buni qilib bo'lmaydi.

Konstanta va o'zgaruvchilardan tashqari ikkala o'zgaruvchi oralig'ida qo'llaniladigan *tiplashgan kostantalar* mavjud. "Tiplashgan" so'zi konstantalarni e'lon qilishda o'zgaruvchilardagi kabi kontantaning tipi ham ko'rsatilishi kerakligini bildiradi.

Yozilishi: CONST

<identifikatori>:<tip>=<qiymat>;

Masalan.

```
  CONST
    VideoSeg : word = $B800;
```

Berilganlarning standart tiplari

Tip, skalyar tiplar, standart va foydalanivchining tiplariga butun sonli, haqiqiy, literli, bulev tipidagi berilganlar, ko'rsatgichlar

Tip - bu qiymatlar to'plami bo'lib, uni dastur ob'ekti qabul qilishi mumkin va shu qiymatlar ustida olib boriladigan operatsiyalar yig'indisidir. Masalan, 1 va 2 soni, butun sonlar tipiga kiradi, ularni qo'shish, ko'paytirish va boshqa arifmetik operatsiyalarni bajarish mumkin. Pascal tilida umumiy holatlarda, tiplarni e'lon qilish uchun TYPE rezervlashgan so'zidan foydalaniladi.

Yozilishi: TYPE

<Tip nomi> = <tip qiymati>;

Berilganlar tipi ikki guruhga bo'linadi: skalyar (oddiy) va strukturalashgan (tarkiblashgan). *Skalyar tiplar* o'z navbatida *standart* va *foydalanuvchi* tiplariga bo'linadi.

Standart skalyar tiplarga *butun, haqiqiy, literli, ko'rsatkich va bulev* tipidagi berilganlar kiradi.

Butun tipdagi berilganlar o'nli yoki o'n oltili sistemalarda berilishi mumkin. Agar son 16 lik sitemada berilgan bo'lsa, uning oldiga $ belgisi quyiladi. 16 lik sistemasidagi sonlarning o'zgarish chegarasi $0000 dan $FFFF gacha.

O'nli sistemadagi sonlar ikki xil usulda yozilishi mumkin:

Qo'zg'aluvchan va qo'zg'almas nuqtali o'zgarmaslar

Haqiqiy o'nli sonlar oddiy arifmetik qoidalarga ko'ra yoziladi. Sonning butun qismi kasr qismidan vergul orqali ajratiladi. Agarda nuqta bo'lmasa, son butun son deb hisoblanadi. Sonning oldiga "+" yoki "-" belgisi quyish mumkin.

Masalan.

125 - butun o'nli son

$1FF - 16 lik son

Qo'zg'aluvchi nuqta ko'rinishidagi haqiqiy butun son quyidagi eksponensial ko'rinishda tasvirlanadi: mE+p, bunda m - mantissa (nuqta bilan ajratilgan butun yoki kasr son), "E" o'nning darajasini bildiradi, p - tartib (butun son).

Masalan.

5.18E+02 = 5.18 * 10^2 = 518

10E-03 = 10 * 10^-3 = 0.01

Foydalaniuvchining tipi - sanaladigan va intervalli dasturchi tomonidan beriladi.

Strukturalashgan tipning asosini bir va bir nechta skalyar tipdagi berilganlar tashkil etadi. Strukturali tiplarga qatorlar, massivlar, to'plamlar, yozuvlar va yangi tipdagi fayl va berilganlar: prosedurali va object tiplar kiradi.

Pascal tilidagi dasturlar protsedura va funksiyalardan tashkil topadi. Dasturning boshida PROGRAM so'zi bilan boshlanuvchi dastur nomi turadi.

Dasturga nom quyish shart emas, lekin dastur nomi bo'yicha axtarilganda uni topish oson bo'ladi, shuning uchun nom quyiladi. Dastur parametrlari standart identifikatori va kiritish-chiqarish Input va Output standart fayllaridan iborat bo'ladi:

PROGRAM PacStat (Input, Output);
PROGRAM MathHandler (Input, Output);

Dastur nomidan keyin 7 bo'limdan iborat bo'lgan dastur bloki keladi: biblioteka modulidagi nomlar ro'yxati (u USES so'zi yordamida aniqlanadi), metkalar yozuvi, konstantalar yozuvi, berilganlar tipini aniqlash, o'zgaruvchilarning yozuvi, protsedura va funksiyalarni yozuvi, operatorlar.

Dastur strukturasi quyidagi ko'rinishga ega:

```
PROGRAM <nom> (Input, Output);
USES <1-nom, 2-nom,...>;
LABEL ...;
CONST ...;
TYPE ...;
VAR ...;
PROCEDURE <nom>;
  <protsedura tanasi>
FUNCTION <nom>;
  < funksiya tanasi>
BEGIN
  <operatorlar>
END.
```

Operator bo'limidan tashqari har qanday boshqa bo'lim qatnashmasligi mumkin. Yozuvlar bo'limi dasturda xohlagan miqdorda qatnashishi mumkin.

USES bo'limi

Bu bo'lim USES so'zidan va standart foydalanuvchi biblioteka modullari nomlari ro'yxatidan iborat bo'ladi.

Yozilishit: USES <1-nom>,<2-nom>,...;

Misol.

```
USES Crt, Dos, MyLib;
```

Metkalarni ifodalash bo'limi

Metka, metkalarni ifodalash bo'limi (Label)

Pascal tilining har qanday operatori oldiga metka quyish mumkin, u shu metkali operatorga goto orqali dasturning xohlagan joyidan to'g'ridan-to'g'ri o'tish mumkinligini ko'rsatadi. *Metka* nom va undan keyin quyiladigan ikki nuqtadan iborat bo'ladi. Nom sifatida son yoki identifikatori qatnashishi mumkin. Metka nomining uzunligi 127 simvolgacha bo'lishi mumkin. Metkadan foydalanishdan oldin, u *metkalarni yozilishi bo'limi*da e'lon qilingan bo'lishi kerak Metkalarni yozilishi bo'limi LABEL (metka) so'zi bilan boshlanadi. Oxirgi nomdan so'ng nuqta vegrul quyiladi.

Yozilishi: LABEL <nom,...>;
Misol.

```
LABEL
  Metka1, Metka2, 111, Blok10;
```

Metka yozilgandan so'ng operatorlar bo'limida ikki nuqta quyiladi:

```
LABEL    M1, M2;  {metkalar yozuvi}
BEGIN
  ...
  M1: <operator>  {M1  ni operatorlar bo'limida ishlatish}
  ...
  M2: <operator>  { M2  ni operatorlar bo'limida ishlatish}
END.
```

metkalarni dastur kengaytmasi bo'yicha ifodalash va qo'llash mukin.

Funksiya va protseduralarni ifodalash bo'limi

Qism dasturi, standart protsedura va funksiyalar,

Bu bo'limda qism dasturlarining tanalari joylashadi. *Qism dasturi deb* dasturning boshqa qismlaridan chaqirilishi mumkin va nomga ega bo'lgan dastur birligiga aytiladi. Pascal dasturlash tilida qism dasturi rolini protsedura va funksiyalar bajaradi. Umumiy hollarda qism dasturi ham dastur kabi strukturaga ega. Qism dasturini ifodalash uchun dasturning boshida yoziladigan PROCEDURE va FUNCTION so'zlaridan foydalaniladi.

Protsedurani yozilishi:

```
PROCEDURE <protsedura nomi  {<parametrlar>};
  <yozuvlar bo'limi >
  <operatorlar bo'limi>
END;
```

Funksiyaning yozilishi:

```
FUNCTION <funksiyaning nomi  {<parametrlar>} : <natija tipi>;
  <yozuvlar bo'limi>
  <operatorlar bo'limi>
END;
```

Protsedura va funksiyalar standart va foydalanuvchi tomonidan aniqlangan bo'lishi mumkin. Standart protsedura va funksiyalr tilning bir qismi bo'lib ularni e'lon qilmasdan ham chaqirish mumkin. Foydalanuvchining protsedura va funksiyalari e'lon qilinishi shart.

Operatorlar bo'limi

Operator, Begin, End.

Pascal tilidagi dasturlarda operatorlar bo'limi asosiy bo'lim hisoblanib, bu bo'limda o'zgaruvchilar, konstantalar, o'zgaruvchilarning qiymatlari e'lon qilinib, ular ustida amallar olib boriladi va natijalar olinadi. Operatorlar

bo'limi *BEGIN* (boshlandi) so'zidan boshlanadi, so'ngra tilning operatorlari yoziladi, ular bir-biridan nuqta vergul orqali ajratiladi.
Bo'lim *END* (tugadi) so'zi bilan tugatiladi va nuqta quyiladi.

```
BEGIN
  <operator;>
  . . .
  <operator>
END.
```

Dasturdagi operatorlar yozilish ketma-ketligi bo'yicha bajariladi.

Izohlar

Izoh, chegara belgilari

Izoh - bu dasturning xohlagan ifodasi bo'lgan tushuntirish matni. Izoh matni () (**) bilan chegaralangan.

Misol.

```
{Regress dasturiga izoh}
(* Lagranj polinomini hisoblash uchun dastur *)
```

(**) chegaralarda bo'sh joy qolishi mumkin emas. Matnda izoh boshlanadigan *chegara belgilari* bo'lmasligi kerak.

Masalan, izoh matni

```
{ Misol {1} vazifa {4} }
```

bu kompilatsiyalash vaqtida xatolikka olib keladi. Lekin () ni (**) ga qo'shib quyish ham mumkin va aksincha

```
(* Misol { 1 }vazifa { 4 } *)
{ Misol (* 1 *) vazifa (* 4 *) }.
```

Foydalanuvchining biblioteka modullari

Biblioteka moduli, biblioteka modulining strukturasi (UNIT, INTERFACE, IMPLEMENTATION)

Biblioteka moduli tushunchasi Turbo Pascal dasturlash tilining idealogiyasida dasturlash sitemasining asosini tashkil etadi. Xuddi shular asosida biblioteka qism dasturlari (protsedura va funksiyalar) tuziladi. Biblioteka moduli- Compile rejimida Destination = Disk direktoriyasi yordamida o'rnatilgan bir yoki bir-nechta protsedura va funksiyalarining kompilyatsiyasi natijasidir. Modul ma'noga ega, u USES bo'limida e'lon qilinadi, va uning yordamida dasturdagi har qanday protsedura yoki funksiyaga murojat qilish mumkin.

Biblioteka modullarini tashkil etishda UNIT, INTERFACE, IMPLEMENTATION, BEGIN, END so'zlariga murojat qilinadi. Sistema kompilyatsiyalanayotgan fayl strukturasini aniqlab, TPU-fayl (agar fayl ichida UNIT va x.k. so'zlari bo'lsa) yoki .EXE-fayl (agar UNIT, IMPLEMENTATION va x.k. lar bo'lmasa) hosil qiladi. Birinchi holatda biblioteka moduli shakllanadi, ikkknchi holatda esa ishga tayyor bo'lgan DOS yo'qlovchi moduli hosil bo'ladi.

*Biblioteka modulining umumiy strukturasi*ni ko'rib chiqamiz:

```
UNIT < biblioteka modulining nomi>;
INTERFACE    {interfeys seksiya}  USES <ulanadigan modul nomi >,...;
    <protsedura sarlavhasi /parametrlari ko'rsatilgan 1-funksiya >
    <protsedura sarlavhasi /parametrlari ko'rsatilgan 2-funksiya >   ...
<protsedura sarlavhasi /parametrlari ko'rsatilgan n-funksiya >
IMPLEMENTATION {ishlatish seksiyasi}
USES <ulanayetgan modulning nomi >,...;
    < parametrlarsiz sarlavha va protsedura tanasi /1-funksiyaning>
    < parametrlarsiz sarlavha va protsedura tanasi /2-funksiyaning>   ...
    < parametrlarsiz sarlavha va protsedura tanasi /n-funksiyaning>
BEGIN      {initsializatsiyalash seksiyasi }   <operator>;   ...
  <operator>
END.
```

Biblioteka modulining nomi diskdagi fayl nomiga mos tushishi kerak. Masalan, agarda fayl Stat.PAS bo'lsa, u holda modulning nomi Stat bo'lishi lozim:

```
UNIT Stat;
```

Initsializatsiya seksiyasi modulning oxirgi seksiyasi hisoblanib, BEGIN va END (modul kod initsializatsiyasiga ega bo'lmasa) so'zlaridan yoki modul initsializatsiyasini bajarishi kerak bo'lgan oprerator qismidan tashkil topadi.

TPU-bibliotekasini tashkil qilishda har bir dasturchida uchraydigan tipik holatni ko'rib chiqamiz. Diskda juda ko'p qo'llaniladigan fodalanuvchining protsedura va funsiyalari saqlanadigan MyLib biblioteka modulini tashkil qilish talab etilsin.

Biblioteka modulini tashkil qilish

1. File bosh menyusi rejimini o'rnatish.
2. Load rejimi yordamida protsedura va funksiyalar matni saqlangan MyLib.PAS faylini yo'qlash (ularda xatolik bor deb faraz qilinadi).
3. Redaktor yordamida biblioteka modulini oluvchi strukturani tashkillashtirish (UNIT, IMPLEMENTATION va x.k. larni qo'llash yordamida).
4. Compile bosh menyu rejimini o'rnatish.
5. Disk holatida Destination qism rejimini o'rnatish.
6. Compile rejimini aktivlashtirib, kompilyatsiyalashni bajarish.
7. Diskda MyLib.TPU biblioteka moduli avtomatik ravishda tashkillashtiriladi.
8. Bibliotekadagi protsedura va funksiyalarning vazifasi, nomi va parametrlari haqidagi malumot beruvchi qisqacha instruksiya yozish.

Biblioteka modullarini qo'llash

1. Xotiraga oldindan tashkil qilingan MyLib biblioteka modulini qo'llash ehtimoli bo'lgan dasturni yo'qlash, masalan., MyProg ni.
2. Bu dasturning USES bo'limida biblioteka modulining nomini ko'rsatish.

```
MyLib:
```

USES MyLib;.

3. Instruksiyaga asosan dasturda MyLib modulidagi kerakli protsedura va funksiyalarni ishlatish.
4. Dasturni yozib bo'lgandan so'ng asosiy menyuga chiqish.
5. Options rejimini o'rnatish.
6. Directories qism rejimida UnitDerictories ni o'rnatib, MyLib moduliga yo'l ko'rsatish.
7. Asosiy menyuga chiqish.
8. Run yordamida MyProg dasturini bajarishga start berish.
9. Dastur ishining to'g'riligiga ishonch hosil qilish.
10. Compile bosh menyusi rejimini o'rnatish.
11. Disk. holatida Destination qism rejimini o'rnatish.
12. Compile rejimini aktivlashtirib, kompilyatsiyalashni bajarish.
13. Diskda dasturning tugallangan maxsuloti bo'lgan, bajariluvchi MyProg.EXE moduli tashkillanadi.

MA'RUZA № 6
MAVZU: CHIZIQLI, TARMOQLANUVCHI VA TAKRORLANUVCHI JARAYONLARNI DASTURLASH.

REJA:

1. Pascal dasturlash tilida kiritish va chiqarish operatorlari.
2. O'zlashtirish operatori.
3. Chiziqli jarayonlarni dasturlash.
4. Paskal tilida shartli va shartsiz o'tish operatorlari.
5. Tanlash operatori.
6. Paskal tilining takrorlash operatorlari.

Kalit so'zlar: Pascal dasturlash tili, operatorlar, sodda operatorlar, kiritish, chiqarish, o'zlashtrish, shartli operatorlar, dasturlash, shartsiz o'tish operatori, protsedurani chiqarish operatori, bo'sh operator, strukturali operator, tanlash operatori, selektor, parametrlar ro'yxati, tanlash o'zgarmaslari ro'yxati.

Dasturda ma'lumotlarning qiymatlarini xotiraga kiritishni bir necha usullarda bajarish mumkin. Sonli o'zgaruvchilarga ularning qiymatini berishda o'zlashtirish operatoridan foydalaniladi. Masalan:

A:=5; V:=6.143;

Dasturni o'zgaruvchilarning turli qiymatlarida bajarish uchun **READ** - kiritish operatori mo'ljallangan.

Kiritish operatori quyidagisha ko'rinishlarda ishlatilishi mumkin:

1) READ(al,a2,...,an);

bunda, al,a2,...,an - o'zgaruvchi qiymatlarini ketma- ket standart INPUT prosedura faylidan oluvshi o'zgaruvchilar. O'zgaruvchilarga qiymatlar turiga mos ravishda klaviaturadan kiritiladi.Aytaylik, A, V, S o'zgaruvchilarga dastur bajarilishi davomida quyidagi qiymatlarni berish kerak bo'lsin:

A=5, V=17, S=6.2.

Operator READ(A,B,S) ko'rinishiga ega bo'lib, sonlar qiymatlarini dastur bajarilishi davomida quyidagisha kiritish mumkin:

5 17 6.2 [Enter].

Agar o'zgaruvchi REAL toifada aniqlangan bo'lsa, uning qiymatini butun son yoki haqiqiy son ko'rinishida kiritiladi. Mashinaning o'zi butun sonni haqiqiy songa o'tkazib oladi.

Masalan:

VAR A, B:REAL;

READ(A, B) operatorining ishlatilishi natijasida 4 va 5 sonlarini probel (bo'sh joy) orqali kiritish mumkin.

2) READLN - bu operator kiritish jarayonida bo'sh qator qoldiradi;

3) READLN(al,a2,...,an);

- operatorning bajarilishida avval al ,a2,...,an ga qiymat kiritilib, so'ng keyingi satrga o'tiladi. Bu operator oldingi ikki operatorga teng kuchlidir.EHM

xotirasidagi ma`lumotlarni displey ekraniga chiqarish operatori - **WRITE** dir. Operator quyidagi bir neshta ko'rinishlarda ishlatilishi mumkin:

1) WRITE(a1,a2, ..., an);

bunda a1,a2,...,an oddiy o'zgaruvchilar, o'zgarmaslar yoki ifodalar bo'lishi mumkin va ular standart OUTPUT prosedura fayliga chiqariladi.

Masalan:

WRITE(`B ning qiymati =', V) operatori displey ekraniga:

V ning qiymati = va undan so'ng V o'zgaruvchining qiymatini chiqaradi.

WRITE operatorida butun va haqiqiy sonlarni ma`lum formatda chiqarish mumkin. Bu format ikki nuqta orqali o'zgaruvchidan so'ng ko'rsatiladi.

Masalan:

WRITE(Y:5:2);

operatori bilan Y ning qiymatini chiqadrishda, Y ning hamma qiymatini chiqarish uchun 5 ta xona ajratilishi, ulardan ikkitasi kasr qismi uzunligini anglatadi (bunda sonning butun va kasr qismini ajratuvshi vergul (nuqta) ham hisobga olinishi zarur). Butun sonlarni chiqarishda kasr qismi formati ko'rsatilmaydi.

Aytaylik, N=179 butun sonli qiymatni chiqarish kerak bo'lsin. Chiqarish operatori buning uchun quyidagi ko'rinishda bo'lishi mumkin:

WRITE (`N=% N:3)

Bu yerda sonni tasvirlash uchun 3 pozitsiya ajratilgan. Agar format 3 dan ortiq berilsa, masalan,

WRITE(`N=',N:5)

bo'lsa, unda sondan oldin ikkita bo'sh joy tashlanadi:

N= ␣␣ 179,

manfiy son uchun e`sa bitta bo'sh joy tashlanadi: N= - 179.

Paskal tilida boshqa chiqarish operatorlari ham ishlatiladi. Parametrlarsiz chiqarish operatori

WRITELN - displey ekranida yangi satrga o'tishni ta`minlaydi.

3)WRITELN(a1,a2,...,an);

- chiqarish operatori oldin a1,a2,...,an larning qiymatlarini chiqaradi, so'ng yangi qatorga o'tishni ta`minlaydi. Shunday qilib, bu ham quyidagi ikki operatorga ekvivalent:

WRITE(a1,a2,...,an); WRITELN;

Masalan, A, V, S qiymatlarini kiritish uchun quyidagi lavhadan foydalanish mumkin:

WRITE(`A, V, S qiymatlarini kiriting');

READ(A, V, S);

Shunday qilib, A, V, S ning qiymatlarini kiritilishidan oldin ekranga quyidagi xabar chiqariladi:

A, V, S qiymatlarini kiritingshundan so'nggina qiymatlarni kiritish mumkin, masalan, 5 17 6.2 [Enter].

O'zlashtirish operatori

Odatda dastur natijasini hosil qilish uchun juda ham ko‘p oraliq hisob ishlarini bajarishga to‘g‘ri keladi. Oraliq natijalarni esa ma’lum muddatga saqlab turish lozim bo‘ladi. Bu ishlarni bajarish uchun tilning eng asosiy operatorlaridan biri bo‘lmish - o‘zlashtirish operatori ishlatiladi:

<o‘zlashtirish operatori>:=<o‘zgaruvchi>:=<ifoda>;

Bu yerda := o‘zlashtirish belgisi hisoblanadi, bu belgini =(tenglik) belgisi bilan almashtirmaslik zarur. O‘zlashtirish operatorida := belgisining o‘ng tomonidagi <ifoda> qiymati aniqlanilib, so‘ng chap tomondagi o‘zgaruvchiga o‘zlashtiriladi yoki boshqacha qilib aytganda, ifoda qiymati o‘zgaruvchi nomi bilan xotirada eslab qolinadi. O‘zgaruvchining oldingi qiymati esa (agar u bo‘lsa) yo‘q bo‘lib ketadi.

O‘zlashtirish operatorini yozishdagi eng muhim narsa, bu ifoda va o‘zgaruvchilarning bir xil turli bo‘lishligidir.

O‘zlashtirish belgisining o‘ng tomonidagi ifodaning natijaviy turiga qarab, o‘zlashtirish operatorini uch xil guruhga ajratish mumkin: arifmetik o‘zlashtirish operatori, mantiqiy o‘zlashtirish operatori, belgili o‘zlashtirish operatori.

Arifmetik o‘zlashtirish operatori

Butun yoki haqiqiy turli, sonli natija beruvchi ifodani (odatda bunday ifodani arifmetik ifoda deb ataladi) hisoblash uchun arifmetik o‘zlashtirish operatoridan foydalaniladi. Arifmetik ifodada qatnashuvchi barcha o‘zgaruvchilar haqiqiy yoki butun turli bo‘lishi kerak. Arifmetik ifoda- sonlar, o‘zgarmaslar, o‘zgaruvchilar va funksiyalardan tashkil topadi, hamda +, -, *, /, div, mod kabi amallar yordamida yoziladi. Arifmetik amallarni bajarilishi quyidagi tartibda bo‘ladi : *, /, *div, mod*, +, .

Ifodani bajarilishidagi bu tartibni o‘zgartirish uchun kichik qavslardan foydalaniladi. Ifodaning qavslar ichiga olib yozilgan qismlari mustaqil holda birinchi galda bajariladi.

Sanab o‘tilgan arifmetik amallarning vazifalari bizga matematika kursidan ma’lum. Lekin, bu ro‘yxatdagi div va mod amallari bilan tanish emasmiz. Div – butun bo‘lishni anglatadi, bo‘linmani butun qismi qoldirilib, qoldiq tashlab yuboriladi. Misol:

7 div 2 = 3
5 div 3 = 1
-7 div 2 = -3
-7 div –2 = 3
2 div 5 = 0
3 div 4 = 0

Mod – butun sonlar bo‘linmasining qoldig‘ini aniqlaydi. m mod n qiymat faqat n>0 dagina aniqlangan. Agar m≥0 bo‘lsa m mod n=m-((m div n)*n), m<0 bo‘lsa m mod n=m-((m div n)*n)+n, m mod n ning natijasi doim musbat sondir.

Misol:

7 mod 2 = 1
3 mod 5 = 3
(-14) mod 3 = 1
(-10) mod 5 = 0

Arifmetik ifodaga doir misollar :
2*5 - 4*3,
9 div 4/2,
45/5/3,
a + b/2*7.2 - sqrt(7),
exp(2 - a)*9.7 - 6.1*6.1

Paskal tilida darajaga ko'tarish amali yo'q, shuning uchun, bu amalni bajarishda logarifmlash qoidasidan foydalanamiz.

Misol: $y=a^n$,$a>0$ ifodani hisoblashni ko'rib chiqaylik. Tenglikni ikkala tomonini logarifmlaymiz:

$Lny=lna^n$, logarifm xossasiga ko'ra

$Lny = nlna$, bu tenglikdan "u" ni aniqlaymiz,

$U=e^{nlna}$ - bu tenglikni Paskal tilida quyidagicha yozish mumkin: *y=exp(n*ln(a)).*

Endi sal murakkabroq arifmetik ifodalarni Paskal tilida yozilishini ko'rib chiqaylik.

Matematik yozuvi	Paskal tilidagi yozuvi
$\frac{a+b}{c+d}$	(a+b)/(c+d)
$\frac{a(a+b)}{bc}$	a*(a+b)/(b*c)
$\frac{1}{1-\frac{1}{1-\frac{1}{x}}}$	1/(1-1/(1-1/x))
$2-(x-b)^2-e^{ax}+\sin CX$	2-sqr(x-b)-exp(a*x)+sin(c*x)
$\frac{x\left(e^{x^2-y^2}-1\right)}{\sqrt{\lvert x^2+y^2\rvert}}$	x*(exp(x*x+y*y)-1)/sqrt(abs(x*x+y*y))

Endi arifmetik o'zlashtirish operatoriga doir misollar ko'rib chiqamiz:

x:= 0;
*c:= sqrt(a*a+b*b);*
*y:= 2*pi*r; i:= i+1; i:= 5.4; x:= a - b/2;*

O'zlashtirish operatorining o'ng tomonidagi ifodada qatnashuvchi o'zgaruvchilar, albatta, bu operatordan oldin o'zining qiymatlariga ega bo'lishi kerak. Aks holda, o'zlashtirish operatori o'z ishini bajara olmaydi. Dastur tuzishda ko'pchilik yo'l qo'yadigan xatolikni quyidagi misolda taxlil qilib ko'ring:

To'g'ri tuzilgan dastur	**Noto'g'ri tuzilgan dastur**
Program Misol; *Var* *a,x,y:Real;* *Begin* *a:=2.3;* *x:=3.1;* ***y:=a*x;*** *Writeln('y=',y);* *End.*	*Program Misol;* *Var* *a,x,y:Real;* *Begin* *a:=2.3;* ***y:=a*x;*** *{o'zlashtirish operatorining o'ng tomonidagi "X" o'zgaruvchining qiymati aniqlanmagan}* *Writeln('y=',y);* *End.*

Mantiqiy o'zlashtirish operatori

Agar o'zlashtirish operatorining chap tomonidagi o'zgaruvchi ***boolean*** (mantiqiy) turiga tegishli bo'lsa, operatorning o'ng tomonida natijasi ***true*** yoki ***false*** bo'lgan mantiqiy ifoda bo'lishi shart.

Mantiqiy ifoda - arifmetik ifoda, solishtirish belgilari va mantiqiy amallardan tashkil topadi. Mantiqiy ifodaning natijaviy qiymati ***true*** (rost) yoki ***false*** (yolg'on) bo'ladi.

Mantiqiy ifodada amallarning bajarilish tartibi quyidagicha:

1. Not
2. *, /, div, mod, and
3. +,-,or
4. =, <, >, <=, >=,<>

Mantiqiy ifodada ham amallar ketma-ketligini o'zgartirish uchun kichik qavslardan foydalaniladi.

Mantiqiy ifodaga doir misollar:

1. x<2*y
2. true
3. not not d
4. (x>y/2)
5. d and (x=y) and b
6. (C or D) and (x=y) or not B

Mantiqiy o'zlashtirish operatoriga doir misollar:

```
d:=true;
b:= (x>y) and (k=0);
c:=D or B and true;
```

```
VAR Global Flag:Boolean;
FUNCTION GETSQR( x:real );
Const  SQRMAX=100;
Begin
    X:=x*x;
    GlobalFlag:=( x>SQRMAX );
    If  GlobalFlag  then  x:=SQRMAX;
    GetSQR:=x;
End;
```

Belgili o'zlashtirish operatori

Agar o'zlashtirish operatorining chap tomonida ***char*** (belgili) yoki String (qatorli) turdagi o'zgaruvchi ko'rsatilgan bo'lsa, u holda operatorning o'ng tomonida belgili ifoda bo'lishi zarur. Belgili qiymatlar ustida faqatgina qo'shish (ulash) amalinigina bajarish mumkin . SHuning uchun, belgili ifoda- belgili o'zgarmas, belgili o'zgaruvchi yoki belgili turli funksiya bo'lishi mumkin.

Belgili o'zlashtirish operatoriga misollar:

```
s:='+-';
d:='*/';
k:=s+d;
p:='Turbo Pascal';
```

```
Program Mkollej;
Var
   s1,s2:String;
Begin
   s1:='oliy';
   s2:=' ta'lim';
   s2:=s1+s2;
   Writeln(s2);
End.
```

Natija:

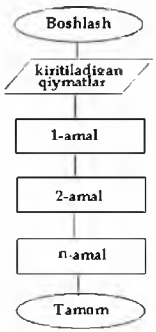
oliy ta'lim

1- misol : Kvadrat tenglamani barcha hollar uchun to'liq hisoblash dasturuni tuzaylik.

$ax^2+bx+c=0$ $D=\sqrt{b^2}-4ac$ $X_{1=}\frac{-b+\sqrt{d}}{2a}$

$X_{2=}\frac{-b-\sqrt{d}}{2a}$

```
Program misol1;
Var a,b,c, x,x1,x2,d:real;
Begin
Read(a,b,c);
If d:=sqrt(b*b-4*a*c);
If d>0 then      begin
X1:=(-b+sqrt(d))/(2*a);
X2:=(-b-sqrt(d))/(2*a);
Writeln ('x1=',x1);
Writeln('x2=',x2);   goto 4;
If d:=0 then
Begin
X1:=-b/2*a;
Writeln('x1=',x1);  end
Else
Writeln( 'yechim yoq')'
4:End.
```



3. Paskal tilida chiziqli jarayonlarni dasturlash

Ta'rif: *Dasturdagi bo'yruqlar navbat bilan ketma-ket bajarilsa bunday dasturlar chiziqli dasturlar deb ataladi.*

Chiziqli tuzilishga ega bo'lgan algoritmlarda ko'rsatmalar yozilish tartibida bajariladi. Ularning blok - sxemasini ishga tushirish, to'xtatish, kiritish-chiqarish jarayoni bloki hamda avvaldan ma'lum jarayon bloklari yordamida tuzilib, bir chiziq bo'ylab ketma-ket joylashgan bo'ladi. Ya`ni hech qanday shart talab qilmaydigan algoritmga chiziqli algoritm deb qarladi. Uni blok sxemasi quydagicha bo'ladi.

Chiziqli tuzilishdagi algoritmni tuzish masalani yechish uchun kerak bo'ladigan boshlang'ich ma'lumotlarni tashkil qiluvchi o`zgaruvchilar nomi, ularning turi va o'zgarish ko'lamini aniqlashdan boshlanadi. Keyin oraliq va yakuniy natijalar o'zgaruvchilarining nomlari, turlari va mumkin bo'lsa, o`zgarish ko'lamini aniqlash kerak. Endi algoritm mana shu boshlang'ich ma'lumotlarni qanday qayta ishlab oraliq va yakuniy natijalarni olish kerakligini aniqlashdan iborat bo'ladi. Buni tushinish uchun turli misollarda aks ettirish.

Misol 1. a va b sonlarining o`rta arifmetigini hisoblash dasturini tuzaylik.

Echish: Demak, $c=\frac{a+b}{2}$

Dasturi:

```
Program urt_arif(input,output);
var a,b: integer;
    begin
        read(a,b);
        c:=(a+b)/2;
        write(c);
    end.
```

Misol 2. Tomonlari mos ravishda a, b, c teng bo'lgan ixtiyoriy ABC uchburchak yuzini hisoblash algoritmini tuzaylik.

Tomonlari ma'lum bo'lganda ABC uchburchakning yuzini topish uchun Geron formulasidan foydalanamiz

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

Bunda

$$p = (a+b+c)/2$$

uchburchakning yarim perimetri.

1. Boshlang'ich ma'lumotlar: *a, b, c* uchburchak tomonlari. Shuning uchun *a, b, c* $\in R$ va $a>0$, $b>0$, $c>0$, ya'ni *a, b, c* — o'zgaruvchilar nomi; ular haqiqiy sonli qiymatlar qabul qiladi. Shuni e'tiborga olish lozimki, bu uchta son uchburchak tomonlarini ifoda qilishi uchun ularning istalgan biri qolgan ikkitasi yig'indisidan katta bo'lmasligi, ya'ni

$$a<b+c,\ b<a+c,\ c<a+b$$

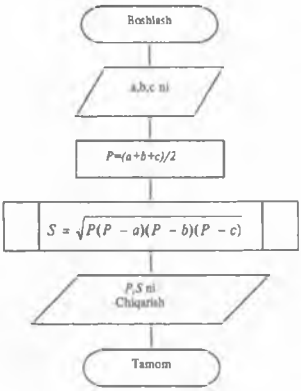
shartlar bajarilishi kerak. Shunday qilib, o'zgarish ko'lami yuqoridagi munosabatlar bilan aniqlanadi.

2. Natijalar: Berilgan formula bilan uchburchak yuzini hisoblash uchun uning yarim perimetrining qiymati kerak. Demak, *p* o'zgaruvchining qiymati oraliq ma'lumot bo'ladi. Yuqoridagi shartlarda $p \in R$ va $p>b$. Yakuniy natija: S — uchburchak yuzi. U $S \in R$ va $S>0$ qiymatlar qabul qiladi.

Shunday qilib, ixtiyoriy ABC uchburchak yuzini EHMda hisoblash va bosmaga (yoki Displey ekraniga) chiqarish

1. *a, b, c* - qiymatlarini EHM xotirasiga kiritish;
2. *p* ning qiymatini formula bilan hisoblash;
3. S ning qiymatini formula bilan hisoblash;
4. *p* va S larning qiymatlarini bosmaga chiqarish operatsiyalaridan iborat bo'ladi. Har qanday algoritmning blok-tarhi ishga tushirish

blokidan boshlanadi. Uni EHMni ishga tayyorlash, boshlang'ich ma'lumotlarni aniqlash va tayyorlash deb tushunish kerak. Hisoblashlarning tugaganligi ana shunday geometrik shakl bilan ko'rsatiladi. Shuning uchun rasmdagi 1 va 6-bloklar ichiga mos kelgan operatsiyalar nomi yozib qo'yilgan.



Boshlang'ich ma'lumotlarni EHMga har xil qurilmalardan kiritish mumkin. Aniq bittasini tanlab olish ish sharoitiga bog'liq. Shuning uchun umumiy kiritish-chiqarish bloklaridan (2- va 5-bloklar) foydalaniladi.

Uchinchi blokda bevosita hisoblash jarayoni, to'rtinchi blokda esa kvadrat ildizdan chiqarish uchun tuzilgan kichik algoritm (yordamchi algoritm) dan foydalanish — avvaldan ma'lum jarayon ko'zda tutilgan. Algoritm ko'r-satmalari yozilish tartibida ketma-ket bajariladi. Ma'lumotlar blokdan blokka yuqoridan pastga uzatiladi. Shuning uchun ularni tutashtiruvchi chiziqqa ko'rsatkichlar qo'yilmagan.

Algoritmdan foydalanuvchi boshlang'ich ma'lumotlarni berilgan shartlar bajariladigan qilib olishi kerak. Aks holda algoritmni bajarib bo'lmaydi. U natijalilik xossasiga ega bo'lmaydi.

Uchburchak yuzini topish dasturini PASKAL tilida tuzamiz.

```
Program uchburchak;
Uses crt;
Var a,b,c,p,S:real;
    Begin
     Write(`a= `); readln(a);
Write('b='); readln(b);
Write('c='); readln(c);
          P:=(a+b+c)/2;
                    S:=sqrt(p*(p-a)*(p-b)*(p-c));
               Writeln(`Uchburchak yuzi S=`,S,` ga teng`);
          End.
```

Dastur kodini kiritib [Ctrl+F9] klavishini bosish bilan dastur ishga tushadi. Natijani ko'rish uchun [Alt+F5] tugmalar kombinatsiyasidan foydalaniladi va ekranda "a= ", "b=", "c=" degan yozuv paydo bo'ladi. Biz uchburchak tashkil qiladigan ixtiyoriy musbat 3 ta sonni kiritamiz va [Enter]

tugmasidan foydalanib natija olamiz. Bu ishlarni paskal muhitida bajarish mumkin.

4. Paskal tilida shartli va shartsiz o'tish operatorlari

Turbo Pascal tilidagi programmaning asosiy qismi operatorlar ketma-ketligidan iborat, har bitta operator berilganlar ustida amal bajaradi. *Operatorlarning buluvchisi* sifatida nuqta vergul belgisi ishlatiladi. Turbo Pascal tilidagi hamma operatorlar ikki guruhga bo'linadi: sodda va strukturali.

Tarkibiga boshqa operatorlar kirmagan operatorlar sodda operatorlar deyiladi. Bunga o'zlashtirish operatori, shartsiz o'tish operatori, protsedurani chiqarish operatori va bo'sh operatorlar kiradi.

O'zlashtirish operatori (:=) o'ng tomonda berilgan ifodani bajarishni va uning qiymatini chap tomonda turgan o'zgaruvchiga tenglashtiradi. Ifoda va o'zgaruvchining tipi bir xil bo'lishi kerak.

Misol:

```
FuncKey := False;
Ch := 'G';
Sum := X + Y;
```

Shartsiz o'tish operatori (goto) "...ga o'tish" degan ma'noni anglatadi va u biror operator bajarilgandan keyin navbatdagi operatorni bajarishga emas balki boshqa biror belgi yordamida belgilangan operatorni bajarishda ishlatiladi. Eslatib o'tamiz, belgi raqam yoki harf simvolidan iborat bo'lishi mumkin.

Misol: GOTO 999;

```
GOTO EndBlock;
```

Goto operatori ishlatilganda belgi ta'sir qiladigan joy bu faqat shu operator yozilgan blok bo'lishi mumkin. Boshqarishni boshqa blokka uzatish man etiladi.

Protseduraning chiqarish operatori foydalanuvchi tomonidan belgilangan protsedurani yoki standart protsedurani ishga tushirish uchun ishlatiladi. Masalan:

```
ClrScr;          {standart protsedurani chiqarish }
InitWork(True);  {foydalanuvchi protsedurasini chiqarish }
```

Bo'sh operator hech qanday amal bajarmaydi va uning tarkibida hech qanday simvollar yo'q. Odatda bo'sh operator lokal yoki global blokning oxiriga o'tishda ishlatiladi:

```
LABEL Metka;
...
BEGIN
  ... GOTO Metka; {blok oxiriga o'tish}
...
Metka:     {bo'sh operatorga belgi bilan murojat qilingan}
END;
```

Murakkab operatorlar

Murakkab operatorlar qat'iyan belgilangan qoidalar bo'yicha boshqa operatorlardan tuzilgan operatorlardir. Hamma strukturali operatorlar uch guruhga bo'linadi:

tarkibiy, shartli, qaytariladigan.

Tarkibiy operator bu bir-biridan nuqta vergul belgisi va BEGIN va END operatorli qavslar yordamida ajratilgan operatorlar guruhidir:

```
BEGIN
  <operator;>
  ...
  <operator>
END;
```

Tarkibiy operator programmaning xohlagan tilning sintaksisi ruxsat beradigan qismida joylashishi mumkin.

Shartli operatorlar.

Shartli operator, shart, joylashtirilgan operatorlar.

Turbo Pascal tilida ikkita shartli operatorlar mavjud: IF va CASE. IF shartli operatori, operatorlarning bajarilish jarayonining tabiiy holatini o'zgartiradigan eng ko'p vositalardan biri. U quyidagi ko'rinishlardan biriga ega bo'lishi mumkin:

```
IF <shart> THEN <operator1>
        ELSE <operator2>;
IF <shart> THEN <operator>;
```

Shart - bu bulev tipdagi ifoda. Birinchi holatda, agar ifoda qiymati haqiqat bo'lsa, <operator1> bajariladi, agar haqiqat emas bo'lsa, <operator2> bajariladi. Ikkinchi holatda - agar ifoda natijasi True bo'lsa, <operator> bajariladi, agar False bo'lsa - IF operatoridan keyingi operator bajariladi. IF operatorlari *joylashtirilgan* bo'lishi mumkin.

Misol:

```
Read(Ch);
  IF Ch='N' THEN Parol:= True  ELSE  Parol:= False;
Read(X);
  IF Parol = True THEN  IF X = 100 THEN Write('Parol va kod to'g'ri.');
ELSE BEGIN
  Writeln ('Kodda xatolik mavjud.');
      Halt(1)
    END.
```

Tanlash operatori. CASE mavjud variantlardan tanlash imkoniyatini beradi. U har biriga tanlash o'zgarmaslari ro'yxati (ro'yxat bitta o'zgarmasdan iborat bo'lishi mumkin) tegishli *selektor* deb nomlangan ifodadan va *parametrlar ro'yxati*dan iborat.

Formati:

```
CASE <ifoda-selektor> OF
  <ro'yxat 1>: <operator 1; >
  <ro'yxat 2>: <operator 2; >
  . . .
  <ro'yxat N>: <operator N>
  ELSE <operator>
END;
```

O'zgarmaslar tipi doim selektor tipiga to'g'ri kelishi kerak. Selektor uchun real va string tiplari man etilgan.

CASE operatori quyidagicha ishlaydi. Birinchi navbatda selektor-ifoda qiymati hisoblanadi, keyingi navbatda joriy selektor qiymatiga teng bo'lgan o'zgarmas qatnashgan operator bajariladi. Agar hech qaysi o'zgarmas selektorning joriy qiymatiga teng bo'lmasa ELSE so'zidan keyingi operator bajariladi. Agar ELSE so'zi bo'lmasa END so'zidan keyingi operator ishga tushadi, ya`ni CASE chegarasidan keyingi operator.

Selektor butun sonli (-32768..32767 diapazonida bo'lgan) bulev, liter yoki foydalanuvchi tipiga bog'liq bo'lishi kerak.

O'zgarmas qiymatlar ro'yxati tasodifiy qiymat yoki diapazondan iborat, ular bir-biridan vergul yordamida ajratiladi. Diapazon chegaralari ikkita biri-biridan ".." belgisi yordamida ajratilgan o'zgaramas sonlar yordamida yoziladi. O'zgarmaslar tipi selektor tipiga to'g'ri kelishi kerak.

Quyida CASE operatorining tipik yozilish tartibi ko'rsatilgan:

Interval tipli selektor :

```
CASE I OF
  1..10: Writeln ('raqam ', I:4, ' diapazon 1 - 10');
  11..20: Writeln ('raqam ', I:4, ' diapazon 11 - 20');
  21..30: Writeln ('raqam ', I:4, ' diapazon 21 - 30')
   ELSE Writeln ('chislo ', I:4, ' kontrolya chegarasidan tashqarida')
END;
```

Butun son tipli selektor :

```
CASE I OF
  1: Z := I + 10;
  2: Z := I + 100;
  3: Z := I + 1000
END;
```

Foydalanuvchi hisob tipli selektor:

```
VAR
  Season: (Winter, Spring, Summer, Autumn);
BEGIN
```

```
... CASE Season OF
  Winter: Writeln('Winter');
  Spring: Writeln('Spring');
  Summer: Writeln('Summer');
  Autumn: Writeln('Autumn')
 END;
END;
```

Paskal tilining takrorlash operatorlari

Yuqorida sanab o'tilgan jarayonlardan biri, takrorlanuvchi jarayonlarni hisoblashni shartli operatorlardan foydalanib ham tashkil etsa bo'ladi, lekin bunday jarayonlarni hisoblashni takrorlash operatorlari yordamida amalga oshirish osonroq kechadi.

Takrorlash operatorlarining 3 xil turi mavjud:

- parametrli takrorlash operatori;
- repeat takrorlash operatori;
- while takrorlash operatori.

Yechilayotgan masalaning mohiyatiga qarab, dastur yozuvchi o'zi uchun qulay bo'lgan takrorlash operatorini tanlab olishi mumkin.

Operatorning quyidagi ko'rinishdagisi amalda ko'proq ishlatiladi:

for k:= k1 *to* k2 *do* S;

bu yerda ***for*** (uchun), ***to*** (gacha), ***do*** (bajarmoq) - xizmatchi so'zlari;

k - sikl parametri (haqiqiy turli bo'lishi mumkin emas);

k1 - sikl parametrining boshlang'ich qiymati;

k2 - sikl parametrining oxirgi qiymati;

S - sikl tanasi.

Operatorning ishlash prinsipi:

- sikl parametri (sp) boshlang'ich qiymat k1 ni qabul qilib, agar bu qiymat k2 dan kichik bo'lsa, shu qiymat uchun S operatori bajariladi;
- sp ning qiymati yangisiga o'zgartirilib (agar k son bo'lsa o'zgarish kadami 1 ga teng, belgili o'zgaruvchi bo'lsa navbatdagi belgini qabul qiladi, va h.k.) yana S operatori bajariladi va bu jarayon k > k2 bo'lguncha davom ettiriladi. SHundan so'ng, sikl operatori o'z ishini tugatib boshqarishni o'zidan keyingi operatorga uzatadi.

Agar biz operatorlarning necha marta takroran hisoblanishini aniq bilsak, u xolda parametrli takrorlash operatoridan foydalanish maqsadga muvofiqdir.

Misol: $\sum_{i=1}^{n} \frac{1}{i}$ yig'indini chekli n ta hadining yig'indisini topish dasturini tuzish.

```
Program sum1;
var
```

```
S: real;
  i,n: byte;  {i va n o'zgaruvchilar 255 dan katta bo'lmagan, butun, natural sonlar}
begin
readln (n); S:= 0;
   for i:=1 to n do
        S:= S + 1/i;
writeln (S);
end.
```

Ayrim paytlarda, sikl parametrini o'sib borish emas, balki kamayish tartibida o'zgartirish mumkin, bu holda sikl operatori quyidagi formada yoziladi:

for k:= k2 ***downto*** k1 *do* S;

bu yerda ***down to*** (gacha kamayib) – tilning xizmatchi so'zi.

Bu operatorda k parametri k2 dan toki k1 gacha kamayish tartibida (agar k - butun qiymatli o'zgaruvchi bo'lsa sikl qadami - 1 ga teng) o'zgaradi. Operatorning ishlash prinsipi oldingi operatornikiday qolaveradi.

Misol. YUqorida ko'rsatilgan misolni dasturini qaytadan tuzaylik. Bu holda dasturdagi sikl operatorigina o'zgaradi xolos:

for i:= n ***downto*** 1 *do*

qolgan operatorlar esa o'z o'rnida o'zgarmay qoladi.

Dasturda parametrli takrorlash operatoridan foydalanish jarayonida, sikl parametrining qiymatini sikl tanasi ichida o'zgartirmaslik lozim, aks holda operatorning ish ritmi buzilishi mumkin. Buni quyidagi misollarda ko'rish mukin:

To'g'ri tuzilgan dastur qismi

```
for i:=1 to 10 do
Begin
s:=i*i;
writeln(s);
end;
```

Noto'g'ri tuzilgan dastur qismi

```
for i:=1 to 10 do
Begin
s:=i*i;
writeln(s);
i:=i+3
end;
```

Ma'lum bir jarayonlarning takrorlash parametrlari haqiqiy qiymatlar qabul qilishi mumkin, bu holda parametrli takrorlash operatoridan to'g'ridan-to'g'ri foydalanib bo'lmaydi. Quyidagi misolda bunday takrorlashlarni qanday tashkil qilish mumkinligini ko'ramiz:

Misol: $y=e^x$ funksiyasini [-2,2] oraliqdagi «x» lar uchun hisoblash dasturini tuzing («x» ning o'zgarish qadami 0,5 ga teng deb hisoblansin).

Funksiyani necha marta hisoblash kerakligini N= formula bilan aniqlaymiz.

```
Program Function;
Var x:real;
```

```
        y:real;
        i:integer;
    begin
    x:=-2;
          for i:=1 to 9 do
          begin
           y:=exp(x);
           writeln(x,y);
           x:=x+0.5
end
end.
```

2. Repeat takrorlash operatori

YUqorida aytib o'tganimizdek, sikldagi takrorlanishlar soni oldindan ma'lum bo'lsa, parametrli (***for***) sikl operatori foydalanish uchun juda qulay. Lekin, ko'pgina hollarda, takrorlanuvchi jarayonlardagi takrorlanishlar soni oldindan ma'lum bo'lmaydi, sikldan chiqish esa ma'lum bir shartning bajarilishi yoki bajarilmasligiga bog'lik holda bo'ladi. Bu hollarda ***repeat*** yoki ***while*** sikl operatorlaridan foydalanish zarur. Agar sikldan chiqish sharti, takrorlanuvchi jarayonning oxirida joylashgan bo'lsa repeat operatoridan, bosh qismida joylashgan bo'lsa ***while*** operatoridan foydalanish maqsadga muvofiqdir.

Repeat operatorining yozilish formasi quyidagicha bo'ladi:

repeat S1; S2; ... SN ***until*** B;

bu yerda ***repeat*** (takrorlamoq), ***until*** (gacha) - xizmatchi so'zlar;

S1, S2, ..., SN lar esa sikl tanasini tashkil etuvchi operatorlar;

B - sikldan chiqish sharti (mantiqiy ifoda).

Operatorning ishlash prinsipi juda sodda, ya'ni siklning tanasi B mantiqiy ifoda rost qiymatli natija bermaguncha takror - takror hisoblanaveradi. Misol sifatida, yana yuqoridagi yig'indi hisoblash misolini olaylik.

```
Program Sum2;
var   i, n: Byte;
      S: real;
begin
   readln(n);
         S:=0; i:=1;
   repeat
     S:=S+1/i;
     i:=i+1;
   until i>n;
   writeln (S)
end.
```

Ayrim takrorlanish jarayonlarida sikldan chiqish shartini ifodalovchi mantiqiy ifoda hech qachon True (rost) qiymatga erishmasligi mumkin. Bu xolda dasturning takrorlash qismi cheksiz marta qaytadan hisoblanishi

mumkin, ya'ni dasturchilar tili bilan aytganda «**dastur osilib qoladi**» shuning uchun, operatordagi shartni tanlashda e'tiborli bo'lish lozim.

E'tiboringizga ya'na bir, ismni qidirib topish dasturini xavola qilamiz:

```
Program BRV;
      Var
  a,b:String[20];
Begin
  a:='Jamshid';
  Repeat
    Writeln('Tanlagan ismingizni kiriting');
    Readln(B);
    if a<>b Then writeln('Noto'g'ri') else writeln('YAshang to'g'ri topdingiz');
  Until A=B;
End.
```

3. While takrorlash operatori

Ahamiyat bergan bo'lsangiz, repeat operatorida siklning tana qismi kamida bir marta hisoblanadi. Lekin, ayrim paytlarda, shu bir marta hisoblash ham yechilayotgan masalaning mohiyatini buzib yuborishi mumkin. Bunday hollarda, quyidagi formada yoziluvchi while sikl operatoridan foydalanish maqsadga muvofiqdir:

while B ***do*** S;

bu yerda ***while*** (hozircha), ***do*** (bajarmoq) - xizmatchi so'zlari;

B - sikldan chiqishni ifodalovchi mantiqiy ifoda;

S - siklning tanasini tashkil etuvchi operator.

Bu operatorda oldin V sharti tekshiriladi, agar u ***false*** (yolg'on) qiymatli natijaga erishsagina sikl o'z ishini tugatadi, aks holda siklni tana qismi qayta - qayta hisoblanaveradi.

While operatoriga misol sifatida, yana yuqorida berilgan yig'indi hisoblash misolini ko'rib chiqaylik:

```
program sum3;
   var   i, n: byte;
         S: real;
   begin
       readln(n);
       i:=1; S:= 0;
       while i<=n do
       begin
           S:= S + 1/i;
           i:= i+1;
       end;
       writeln (S)
end.
```

4. Bo'sh operator

Bu operator o'zidan keyingi operatorni aniqlab beradi xolos. Operatorlar ketma-ketligi orasida boshqa operatorlardan ";" belgisi bilan ajratilib turiladi. Bundan tashqari, bo'sh operator metka bilan jixozlangan ham bo'lishi mumkin.
Misol:

1. **begin** L1:; k:=5; M:=k+6; **end**.
2. **begin** M:=5; k:=M-2.7; L4: **end**.

Ayrim paytlarda, ba'zi bir operatorlarga bir nechta metka bilan murojaat qilishga to'g'ri kelganda bo'sh operatordan foydalanish qo'l keladi.

S5:; S6:; S7: x:=0.5;

Punktuatsiya qoidalari

Operatorlarni yozishda quyidagi punktuatsiya qoidalariga roiya qilish kerak:

1. Nuqta vergul belgisi UNIT, USES, LABEL, TYPE, CONST, VAR rezervlangan so'zlaridan keyin quyilmaydi va har bir opisaniyedan keyin quyiladi.
2. Nuqta vergul belgisi BEGIN so'zidan keyin va END so'zidan oldin quyilmaydi, chunki bu so'zlar operator emas, balki operator qavslaridir.
3. Nuqta vergul operatorlarni ajratishi uchun xizmat qiladi, agar u operatorlar orasida quyilmasa kompilyatsion xato vujudga keladi.
4. Sikl operatorlarida nuqta vergul belgisi WHILE, REPEAT, DO lardan keyin va UNTIL dan oldin qo'yilmaydi.
5. Shartli operatorlarda nuqta vergul THEN dan keyin va ELSE dan oldin quyilmaydi.

MA'RUZA № 7

MAVZU: PASKAL DASTURLASH TILIDA MASSIVLAR BILAN ISHLASH.

REJA:

1. Paskal dasturlash tilida massivlar bilan ishlash.
2. Bir o'lchovli massivlar.
3. Ikki o'lchovli massivlar va ularga doir masalalar.

Paskal dasturlash tilida massivlar bilan ishlash.

Umumiy nomga ega bo'lgan bir xil ko'rinishda tartiblangan elementlar ketma-ketligi massiv deyiladi. Massiv elementlari uning komponentlari deb ataladi. Komponenta tipi – baza tipi hisoblanadi. Har bir tip o'zining indeksiga va nomiga ega,ular qavs ichiga keltiriladi.

Massiv elementlari ixtiyoriy tipda ,hattoki ma'lumotlar ham bo'lishi mumkin. Massiv elementlarining tiplari bazali deyiladi. Massiv elementlarining soni programma ishlash jarayonida o'nga ozlashtirib boriladi. Uning har bir alohida elementiga murojaat massiv elementlariga mos kelgan indekslari bo'yicha bo'ladi. Massiv indeksi tushunchasi xuddi vektorlar indeksi tushunchasi kabi bo'ladi Massivlarni e'lon qilish uchun **Array** of (massivda) so'z birligi ishlariladi.

Yozilishi:

```
TYPE
<tip nomi>=array[indeks tipi] of <komponenta tipi>;
VAR
<identifikator,...>:<tip nomi>;
```

Massivlarni tiplarni e'lon qilmasdan turib ham qo'llash mumkin.

```
VAR
<>identifikator,...:array[indeks tipi] of <komponenta tipi>;
```

Misol;

```
TYPE
Klass=(k1,k2,k3,k4);
Znak=array[1..255] of char;
VAR
M1: Znak;{Znak tipi tiplar bo'limida oldindan keltirilgan}
M2: array[1..60] of integer;{M2 massivning yozilishi}
M3: array[1..4] OF Klass;
Mas: array[1..4] of integer;
```

Massivning baza tipi har qanday tip bo;lishi mumkinligi sababli u boshqa massiv ham bo'lishi mumkin.Natijada,Ko'p o'chamli massiv hosil bo'ladi.

Masalan.

```
TYPE
Vector=array[1..4] of integer;
Massiv=array[1..4] of vetor;
VAR
Matr : Massiv;
```

Xuddi shu strukturani boshqa turdagi yozuvni qo'llash natijasida ham hosil qilish mumkin.:

VAR

Matr: array[1..4,1..4] of integer;

Massiv bilan ishlashda konstantalar ham ishlatilishi mumkin.

CONST

G1=4 ; G2=6;

VAR

MasY : array[1..G1,1..G2] of real;

Massiv elementlari xotirada ketma-ket joylashadi. Indekslari kichkina bo'lgan elementlar xotoraning pastki indekslarida saqlanadi.Ko'p elementli massivlarda eng o'ng tarafidagi indeks birinchi bo'lib o'sib boradi.Masalan ; Agarda

A:array[1..5,1..5] of integer;

Bo'lsa u holda massiv elementlari adreslarnig o'sishi bo'yicha joylashadi.

A[1,1]

A[1,2]

.....

A[1,5]

A[2,1]

A[2,2]

Massivlarni bir butun holda ishlatilgan paytda massiv nomlaridagi indekslardagi kvadrat qavsga olinmay ishlatiladi.Massivlar kiritish operatorida "teng" yoki "teng emas" operatsiyalarida ishlatilishi mumkin

Bu amallarda ishlatiladigan massivlar bir xil tipdagi indeks va komponentalarda ega bo'lib strukturasi bir biriga o'xshash bo'lishi kerak.

Masalan,

A va **B** massivlar **Var A,B: array[1..20] of real;**

Ko'rinishida ifodalangan bo'lsa unda natija quyidagicha bo'ladi.

A=B **True**,agarda **A** massivning elementlarining qiymatlari V massiv elementlarining qiymatlariga mos ravishda teng bo'lsa.

A<>V True, agarda **A** massiv elementining bron bir qiymati V massiv elementining qiymatlariga mos ravishda mos bo'lsa.

A :=V **V** massiv elementining hamma qiymatlari **A** massiv elementlari tomonidan o'zlashtirilsa **V** massiv elementlarining qiymatlari o'zgarmasdan qoladi.

Massiv e'lon qilingandan keyin uni elementlarini nomlari qavs ichida ko'rsatilgan holda ishlatish mumkin..Masalan:**mas[2],vektorZ[10]** massivning ikkinchi va uchinchi elementlariga murojaat etishni bildiradi.

Ikki o'lchovli massivlar ikita indeks n o'lchovlilarida esa n ta indeks ko'rsatiladi

Masalan, **MatrU[4,4]** bu yozuv **MatrU** massivning 4 ta qator 4 ta ustundagi elementini bildiradi.Array tipidagi qiymatlar bilan ishlashda quyidagi holatlar bo'lishi mumkin:

VAR
A,D : array[1..4] of real;
B:array[1..10,1..15] of integer;
I,J: integer;K:integer;S:real;
Bu operatsiyani FOR operatori yordamida ham bajarish mumkin:
FOR I: =1TO 4 Do A[I]:=0;
Ikki o'lchovli massivlarga indeks qo'yish uchun ichma-ich joylashgan operatorlar ishlatiladi:

FOR I:=1TO 10 DO
FOR J:=1 TO 15 DO
B[I,J]:=0;

Paskal algoritmik tilida massiv elementlarini birdaniga kiritish-chiqarish imkoniyati yo'qligi sababli elementlar bittadan kiritiladi.Massiv elementiga qiymatni o'zlashtirish operatori yodamida beriladi,initsializatsiya misolda ko'rsatilgandek ,lekin ko'p hollarda **READ** yoki **READLN** operatori yordamida o'zlashtiriladi va sikl operatoridan foydalanamiz.
FOR I:= 1 TO 10 Do
FOR J:=1 TO 15 DO
READLN(D[I,J]);
WRITELN operatori ishlatilishi sababli har bir qiymat yangi qatordan kiritiladi.Alohida elementlarni qiymatlarini ham kiritish mumkin.Bunda quyidagi operatorlardan foydalaniladi:
READ (A[3]);
READ (B[6,9]);
Bu yerda A vektoriga 3ta element qiymati va V matritsaning 6- qator 9-ustuniga joylashgan elementlari qiymatlarini kiritadi.

Ikkala qiymat ham ekranning bitta qatorida ,kursirning joriy pazitsiyasidan teriladi. Massiv elementlari qiymatini chiqarish ham xuddi shunday bajariladi, lekin bunda WRITE yoki WRITELN operatori qo'llaniladi.
FOR I:=1 TO 4 DO
Writeln (A[I]); {A massiv elementlari qiymatini chiqarish}
Yoki
FOR I:=1 TO 10 DO
FOR J:=1 TO 15 DO
Writeln (V[I,J]); {B massiv elementlari qiymatini chiqarish}
1-Misol:
5 ta elementdan hosil bo'lgan X massivning elementlarini kiriting. Har bir elementning kvadrati va ildizini ekranga chiqaring.
Yechish:

```
PROGRAM P7_2;
VAR
Sum,I:integer;
Sr:real;
X:array [1..5] of integer;
Kor,Kv:array[1..5] of real;
Begin
Sum:=0;
```

```
Writeln('5 ta butun qiymatlarni kiriting:');
FOR I:=1 TO 5 DO BEGIN
Write('1-sonni kiriting: ');
Readln (X[I]);
Kor[I]:= sqrt(X[I]);
Kv[I]:=sqr(X[I]);
END;
FOR I:=1 TO 5 DO
Write(Kor[I]:8:2);
Writeln;
FOR I:=1 TO 5DO Write (Kv[I]:8:2);
Readln;
END.
```

2-misol:

3x3 massiv elementlarini kiriting va har bir qator yig'indisini hisoblang.

Yechish:

```
PROGRAMP7_8;
VAR
Xarray[1..3,1..3] of integer;
Sum:array[1..3] of integer;
I,J:integer;
BEGIN
Randomize;
FOR I=1 TO 3 DO
FOR J:=1 TO 3 DO BEGIN
X[I,J]:=random(300);
END;
FOR I:=1 TO 3DO
Sum[I]:=0
FOR I:=1 TO 3 DO
FOR J:=1TO 3 DO
Sum[I]:=Sum[I]+X[I,J];
FOR I:=1 TO 3 DO
Write(Sum[I]:3);
Readln;
END.
```

Massivdan nusxa olish deb, bitta massiv elementlari qiymatlarini boshqa massiv elementlarini o'zlashtirishiga aytiladi. Nusxa olish bitta o'zlashtirish operatori yordamida amalga oshirish mumkin,masalan **A:=D** yoking **FOR** operatori yordamida .

FOR I: =1TO 4 DO A[I]:=D[I];

Ikkala holda ham massiv elementlari qiymatlari o'zgarmaydi, A massiv elementlarining qiymatlari D massiv elementlari qiymatiga teng bo'lib qoladi. Ko'rinib turibdiki ,ikkala massiv ham strukturasi bo'yicha bir-biriga o'xshash. Ko'pchilik holatlarda massivda qaysidir elementlarni izlashga to'g'ri keladi. Masalan: A massivning nechta elementi nol qiymatga ega ekanligini bilish talab etiladi. Buning uchun qo'shimcha o'zgaruvchi K ni kiritamiz va FOR , IF operatorlarida foydalanamiz.

```
K:=0;
FOR I:=1 TO 4 DO
IF A[I]=0 THEN K:=K+1;
```

Sikl bajarilgandan keyin K o'zgaruvchi A massivning nolga teng bo'lgan qiymatlarini o'z ichiga oladi. Massiv elementlari qiymatlarini joyini almashtirish massivning bazali tipiga o'xshash tipdagi yordamchi o'zgaruvchi yordamida amalga oshiriladi.Masalan, A massivning birinchi va beshinchi elementlari qiymatlarini joyini almashtiring.

```
Vs:=A[5]
A[5]:=A[1];
A[1]:=Vs;
```