Reference Guide to
Turbo Pascal Programs

ArFcnTab

Function Constructs a TABle of values of the six ARithmetic FunCtioNs \( \omega(n) = \sum_{p\mid n} 1, \ \Omega(n) = \sum_{p\mid n} a, \ \mu(n), \ d(n) = \sum_{d\mid n} 1, \ \phi(n), \ and \ \sigma(n) = \sum_{d\mid n} d. \)

Syntax arfcntab

Commands

- PgUp Display the preceding 20 values
- PgDn Display the next 20 values
- J Jump to a new point in the table
- P Print 500 values, starting at the top of the displayed screen
- Esc Escape from the environment

Restrictions \( 1 \leq n < 10^9 \)

Algorithm When the program begins execution, it first constructs a list of the primes not exceeding \( 10^{9/2} \), by sieving. These primes are used for trial division. The factorizations are determined simultaneously for all 20 numbers (or all 500 numbers, in the case of printing).

See also Pi

Car

Function Computes the CARmichael function \( c(m) \), which is defined to be the least positive integer \( c \) such that \( a^c \equiv 1 \pmod{m} \) whenever \( (a, m) = 1 \).

Syntax car [m]

Restrictions \( 1 \leq m < 10^{18} \)

Algorithm First the canonical factorization of \( m \) is determined by trial division. If \( p \) is an odd prime then \( c(p^j) = p^{j-1}(p - 1) \). Also, \( c(2) = 1, \ c(4) = 2, \) and \( c(2^j) = 2^{j-2} \) for \( j \geq 3 \). Finally, \( c(m) \) is the least common multiple of the numbers \( c(p^a) \) for \( p^a \mid m \).
ClaNoTab

Function
Constructs a TABLE of CLASS Numbers of positive definite binary quadratic forms. The number $H(d)$ is the total number of equivalence classes of such forms of discriminant $d$, while $h(d)$ counts only those equivalence classes consisting of primitive forms.

Syntax
clanotab

Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgUp</td>
<td>Display the preceding 40 values</td>
</tr>
<tr>
<td>PgDn</td>
<td>Display the next 40 values</td>
</tr>
<tr>
<td>J</td>
<td>Jump to a new point in the table</td>
</tr>
<tr>
<td>P</td>
<td>Print $h(d)$ and $H(d)$ for $-2400 \leq d &lt; 0$</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

Restrictions
$-10^4 \leq d < 0$

Algorithm
All reduced triples $(a, b, c)$ are found, with $0 < a < \sqrt{10^4/3}$. When a reduced triple is located, the value $d = b^2 - 4ac$ is calculated, and the count of $H(d)$ is increased by 1. If $\gcd(a, b, c) = 1$ then the count of $h(d)$ is also increased by 1. The entire table is calculated before the first screen of values appears. This may take several minutes on a slow machine.

See also
QFormTab, Reduce

Comments
The time required to calculate class numbers in this manner in the range $-D \leq d < 0$ is roughly proportional to $D^{5/2}$, and roughly $D$ numbers must be stored. By adopting a more sophisticated algorithm, one could calculate only those values that are to appear on a given screenful, and the time required for the calculation would be much smaller, making it feasible to construct a program of this sort that would accommodate $d$ in the range $-10^9 \leq d < 0$, say. For faster algorithms, see D. Shanks, *Class number, a theory of factorization, and genera*, Proc. Sympos. Pure Math. 20, Amer. Math. Soc. Providence, 1970, 415–440. For a method that is theoretically still faster, but that may be challenging to implement, see J. L. Hafner and K. S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, J. Amer. Math. Soc. 2 (1989), 837–850.

CngArTab

Function
Displays the addition and multiplication TABLEs for CoNGruence ARithmetic $(\mbox{mod } m)$.
Syntax  cngartab

Commands

↑      Move up
↓      Move down
←      Move left
→      Move right
a      Start at column a
b      Start at row b
m      Set modulus m
s      Switch between addition and multiplication
r      Display only reduced residues (in multiplication table)
p      Print the table (if \( m \leq 24 \))
Esc    Escape from the environment

Restrictions  \( 1 \leq m < 10^9 \)

See also  PowerTab

---

CRT

Function  Determines the intersection of two arithmetic progressions. Let \( g = (m_1, m_2) \). The set of \( x \) such that \( x \equiv a_1 \pmod{m_1} \), \( x \equiv a_2 \pmod{m_2} \) is empty if \( a_1 \not\equiv a_2 \pmod{g} \). Otherwise the intersection is an arithmetic progression \( a \pmod{m} \). In the Chinese Remainder Theorem it is required that \( g = 1 \), and then \( m = m_1m_2 \). In general, \( m = m_1m_2/g \).

Syntax  
\[
crt [a_1 \ m_1 \ a_2 \ m_2]
\]

Restrictions  \( |a_i| < 10^{18} \), \( 1 \leq m_i < 10^{18} \)

Algorithm  First the linear congruence \( m_1y \equiv a_2 - a_1 \pmod{m_2} \) is solved. If \( a_1 \not\equiv a_2 \pmod{g} \), then this congruence has no solution, and the intersection of the two given arithmetic progressions is empty. Otherwise, let \( y \) denote the unique solution of this congruence in the interval \( 0 \leq y < m_2/g \). Then the intersection of the two given arithmetic progressions is the set of integers \( x \equiv a \pmod{m} \) where \( a = ym_1 + a_1 \) and \( m = m_1m_2/g \).

See also  CRTDem, IntAPTab, LinCon, LnCnDem

Comments  This program provides a user interface for the procedure CRThm found in the NoThy unit. To see how the algorithm is implemented, examine the file nothy.pas.

---

CRTDem

Function  Demonstrates the method employed to determine the intersection of two given arithmetic progressions.
**Syntax**
crtdem [a1 m1 a2 m2]

**Restrictions**
\[|a_i| < 10^{18}, \ 1 \leq m_i < 10^{18}\]

**Algorithm**
See the description given for the program CRT.

**See also**
CRT, IntAPTab, LnCnDem

---

**DetDem**

**Function**
Demonstrates the method used to evaluate \(\det(A) \mod m\).

**Syntax**
detdem

**Restrictions**
\(0 < m < 10^9\), \(A = [a_{ij}]\) is \(n \times n\) with \(1 \leq n \leq 9\), \(|a_{ij}| < 10^9\)

**Algorithm**
See description for the program DetModM.

**See also**
DetModM, SimLinDE

---

**DetModM**

**Function**
Determines \(\det(A) \mod m\).

**Syntax**
detmodm

**Commands**

- A Assign dimension of matrix
- B Build matrix
- C Choose modulus
- D Determine value of \(\det(A) \mod m\)
- E Exit
- F Form altered matrix

**Restrictions**
\(0 < m < 10^9\), \(A = [a_{ij}]\) is \(n \times n\) with \(1 \leq n \leq 9\), \(|a_{ij}| < 10^9\)

**Algorithm**
Row operations are performed until the matrix is upper-triangular. After each row operation, the elements of the new matrix are reduced modulo \(m\). The row operations used are of the following two types: (i) Exchange two rows (which multiplies the determinant by \(-1\)); (ii) Add an integral multiple of one row to a different row (which leaves the determinant unchanged).

**See also**
DetDem, SimLinDE

**Comments**
This program provides a user interface for the function DetModM, which is defined in the file det.i.

---

**EuAlDem1**

**Function**
Demonstrates the calculation of \((b,c)\) by using the identities \((b,c) = (-b,c), (b,c) = (c,b), (b,c) = (b + mc,c), (b,0) = |b|\).

72 Reference Guide to Turbo Pascal Programs
Algorithm
The number $m$ is chosen so that $b + mc$ lies between $0$ and $c$. The systematic use of the Division Algorithm in this way is known as the Euclidean Algorithm.

See also
EuAlDem2, EuAlDem3, FastGCD, GCD, GCDTab, LnComTab, SlowGCD

EuAlDem2
Function
Demonstrates the extended EUclidean ALgorithm by exhibiting a table of the quotients $q_i$, remainders $r_i$, and the coefficients $x_i$, $y_i$ in the relations $r_i = x_ib + y_ic$.

Syntax
eualdem2

Commands
- PgUp: Display the top portion of the table
- PgDn: Display the bottom portion of the table
- b: Enter a new value of $b$
- c: Enter a new value of $c$
- P: Print the table
- Esc: Escape from the environment

Restrictions
$0 < b < 10^{18}$, $0 < c < 10^{18}$

See also
EuAlDem1, EuAlDem3, FastGCD, GCD, GCDTab, LnComTab, SlowGCD

EuAlDem3
Function
Demonstrates the extended EUclidean ALgorithm in the same manner as EuAlDem2, but with rounding to the nearest integer instead of rounding down.

Syntax
eualdem3

Commands
- PgUp: Display the top portion of the table
- PgDn: Display the bottom portion of the table
- b: Enter a new value of $b$
- c: Enter a new value of $c$
- P: Print the table
- Esc: Escape from the environment

Restrictions
$0 < b < 10^{18}$, $0 < c < 10^{18}$
See also EuAlDem1, EuAlDem2, FastGCD, GCD, GCDTab, LnComTab, SlowGCD

---

**FacTab**

**Function** Constructs a TABle of the least prime FACTor of odd integers from $10N + 1$ to $10N + 199$.

**Syntax** `factab`

**Commands**

- **PgUp** Display the preceding 100 values (i.e. decrease $N$ by 20)
- **PgDn** Display the next 100 values (i.e. increase $N$ by 20)
- **N** New $N$; view table starting at $10N + 1$
- **Esc** Escape from the environment

**Restrictions** Integers not exceeding $10^9 + 189$ (i.e. $0 \leq N \leq 99999999$).

**Algorithm** When the program begins execution, it first constructs a list of the odd primes not exceeding $\sqrt{10N + 200}$, by sieving. We call these the “small primes.” There are 15803 such primes, the last one being 31607. The next prime after this is 31621. When $N$ is specified, the odd integers in the interval $[10N, 10N + 200]$ are sieved by those small primes not exceeding $\sqrt{10N + 200}$; least prime factors are noted as they are found.

**See also** Factor, GetNextP

---

**Factor**

**Function** FACTORs a given integer $n$.

**Syntax** `factor [n]`

**Restrictions** $|n| < 10^{18}$

**Algorithm** Trial division. After powers of 2, 3, and 5 are removed, the trial divisors are reduced residues modulo 30.

**See also** P-1, P-1Dem, Rho, RhoDem

**Comments** Factors are reported as they are found. The program can be interrupted by touching a key. This program provides a user interface for the procedure Canonic found in the NoThy unit. To view the source code, examine the file `nothy.pas`.

---

**FareyTab**

**Function** Constructs a TABle of FAREY fractions of order $Q$. Fractions are displayed in both rational and decimal form, up to 20 of them at a time.
Syntax  fareytab

Commands  
- **PgUp**: View the next 19 smaller entries
- **PgDn**: View the next 19 larger entries
- **D**: Center the display at a decimal x
- **R**: Center the display at a rational number a/q
- **P**: Print the table (allowed for Q ≤ 46)
- **Esc**: Escape from the environment

Restrictions  1 ≤ Q < 10^9

Algorithm  If a/q and a’/q’ are neighboring Farey fractions of some order Q, say a/q < a’/q’, then a’q – q’a = 1. By the extended Euclidean algorithm, for given relatively prime a and q we find x and y such that xq – ya = 1. Then q’ = y + kq, a’ = x + ka where k is the largest integer such that y + kq ≤ Q. With a/q given, the next smaller Farey fraction a’/q’ is found similarly. The Farey fractions surrounding a given decimal number x are found by the continued fraction algorithm. Fractions are computed only as needed by the screen or the printer.

FastGCD

Function  Times the execution of the Euclidean algorithm in calculating the Greatest Common Divisor of two given integers.

Syntax  fastgcd

Restrictions  |b| < 10^{18}, |c| < 10^{18}

Algorithm  Euclidean algorithm, rounding down.

See also  GCD, SlowGCD

FctrlTab

Function  Provides a table of n! (mod m). Each screen displays 100 values.

Syntax  fctrltab

Commands  
- **PgUp**: View the preceding 100 entries
- **PgDn**: View the next 100 entries
- **J**: Jump to a new position in the table
- **M**: Enter a new modulus
- **P**: Print the first 60 lines of the table
- **Esc**: Escape from the environment

Restrictions  0 ≤ n ≤ 10089, 0 < m < 10^6

Reference Guide to Turbo Pascal Programs  75
Algorithm

All 10089 values are calculated as soon as \( m \) is specified, unless \( m < 10089 \), in which case only \( m \) values are calculated.

---

**GCD**

**Function**
Calculates the Greatest Common Divisors of two given integers.

**Syntax**
\[
gcd \ [b \ c]
\]

**Restrictions**
\(|b| < 10^{18}, |c| < 10^{18}\)

**Algorithm**
Euclidean algorithm with rounding to the nearest integer.

**See also**
EuAlDem1, EuAlDem2, EuAlDem3, FastGCD, GCDTab, LnComTab, SlowGCD

**Comments**
This program provides a user interface for the function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

**GCDTab**

**Function**
Displays \((b, c)\) for pairs of integers.

**Syntax**
\[
gcdtab
\]

**Commands**
\[
\uparrow \quad \text{Move up} \\
\downarrow \quad \text{Move down} \\
\leftarrow \quad \text{Move left} \\
\rightarrow \quad \text{Move right} \\
b \quad \text{Center table on column } b \\
c \quad \text{Center table on row } c \\
\text{Esc} \quad \text{Escape from the environment}
\]

**Restrictions**
\(|b| < 10^{18}, |c| < 10^{18}\)

**Algorithm**
Euclidean algorithm.

**See also**
GCD, EuAlDem1, EuAlDem2, EuAlDem3, LnComTab

---

**GetNextP**

**Function**
Finds the least Prime larger than a given integer \( x \), if \( x \leq 10^9 \). If \( 10^9 < x < 10^{18} \), it finds an integer \( n \), \( n > x \), such that the interval \((x, n)\) contains no prime but \( n \) is a strong probable prime to bases 2, 3, 5, 7, and 11. A rigorous proof of the primality of \( n \) can be obtained by using the program ProveP.

Reference Guide to Turbo Pascal Programs
**Syntax**
getnextp [x]

**Restrictions**
0 ≤ x < 10^{18}

**Algorithm**
If 0 ≤ x ≤ 10^9 then the least prime larger than x is found by sieving. If 10^9 < x < 10^{18} then strong probable primality tests are performed.

**See also**
FacTab, ProveP

**Comments**
For 0 ≤ x ≤ 10^9, this program provides a user interface for the function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas. For 10^9 < x < 10^{18} this program uses the function SPsP, which is found in the unit NoThy, with source code in the file nothy.pas.

---

**Hensel**

**Function**
Provides a table of solutions of \( f(x) \equiv 0 \pmod{p^j} \), in the manner of HENSEL’s lemma. All roots \((\text{mod } p)\) are found, by trying every residue class. If \( f(a) \equiv 0 \pmod{p} \) and \( f'(a) \not\equiv 0 \pmod{p} \), then a tower of roots lying above \( a \) is displayed. If \( f'(a) \equiv 0 \pmod{p} \) then roots lying above \( a \) are exhibited only one at a time. Roots \((\text{mod } p^j)\) are displayed both in decimal notation and in base \( p, a = \sum_{i \geq 1} c_i p^{i-1} \). The user must choose between viewing singular or non-singular roots. The display starts with a non-singular root, if there are any.

**Syntax**
hensel

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Lift to larger values of ( j )</td>
</tr>
<tr>
<td>↓</td>
<td>Drop to smaller values of ( j )</td>
</tr>
<tr>
<td>←</td>
<td>Shift left in the table</td>
</tr>
<tr>
<td>→</td>
<td>Shift right in the table</td>
</tr>
<tr>
<td>S</td>
<td>Switch to singular roots</td>
</tr>
<tr>
<td>N</td>
<td>Switch to non-singular roots</td>
</tr>
<tr>
<td>D</td>
<td>Define the polynomial</td>
</tr>
<tr>
<td>p</td>
<td>Choose the prime modulus</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

**Restrictions**
2 ≤ p < 2000, \( p^j \leq 10^{18} \), \( f(x) \) must be the sum of at most 20 monomials

**Algorithm**
The polynomial \( f(x) \) is evaluated at every residue class, and an array is formed of the roots. For each root found, the quantity \( f'(x) \) is calculated, in order to determine whether the root is singular or not.

**See also**
PolySolv

*Reference Guide to Turbo Pascal Programs* 77
**HSortDem**

**Function**  
DEMonstrates the HeapSORT algorithm of J. W. J. Williams, by applying the algorithm to \( n \) randomly chosen integers taken from the interval \([0, 99]\). This algorithm is employed in the programs Ind and IndDem.

**Syntax**  
`hsortdem`

**Restrictions**  
\( 1 \leq n \leq 31 \)

---

**Ind**

**Function**  
Given \( g, a, \) and \( p, \) finds the least non-negative \( \nu \) such that \( g^\nu \equiv a \pmod{p} \), if such a \( \nu \) exists. Thus, if \( g \) is a primitive root of \( p, \) then \( \nu = \text{ind}_g a. \)

**Syntax**  
`ind [g a p]`

**Restrictions**  
\(|g| < 10^9, |a| < 10^9, 1 < p < 10^9, (g, p) = 1\)

**Algorithm**  
First LinCon is used to find \( \overline{\pi} \pmod{p} \) so that \( g^\overline{\pi} \equiv 1 \pmod{p} \). The number \( s \) is taken to be either the integer nearest \( \sqrt{p} \) or else 10000, which ever is smaller. A table is made of the residue classes \( a g^j \pmod{p} \) for \( 0 \leq j < s \). This table is sorted by the HeapSort algorithm into increasing order. For \( j = 0, 1, \ldots, \) a search is conducted (by binary subdivisions) to see whether the residue class \( g^{j s} \pmod{p} \) is in the table. If a match is found, then \( \nu = is + j \). If \( j \) reaches \( p/s \) without finding a match, then \( a \) is not a power of \( g \pmod{p} \). Thus the index is found in time \( O(p^{1/2} \log p) \). This method was suggested by D. Shanks.

**See also**  
IndDem, IndTab, Power, PowerTab

---

**IndDem**

**Function**  
DEMonstrates procedure used to compute \( \text{ind}_g a \pmod{p} \).

**Syntax**  
`inddem [g a p]`

**Restrictions**  
\(|g| < 10^9, |a| < 10^9, 1 < p < 10^9\)

**Algorithm**  
See the description of the program Ind.

**See also**  
Ind, IndTab, Power, PowerTab

---

**IndTab**

**Function**  
Generates a TABle of INDices of reduced residue classes modulo a prime number \( p, \) with respect to a specified primitive root. Also generates a
table of powers of the primitive root, modulo $p$. Up to 200 values are displayed a one time.

**Syntax**

indtab

**Commands**

- PgUp View the preceding 200 entries
- PgDn View the next 200 entries
- J Jump to a new position in the table
- E Switch from indices to exponentials
- I Switch from exponentials to indices
- M Enter a new prime modulus
- B Choose a new primitive root to use as the base
- P Print table(s)
- Esc Escape from the environment

**Restrictions**

$p < 10^4$

**Algorithm**

The least positive primitive root $g$ of $p$ is found using the program PrimRoot. The powers of $g$ modulo $p$ and the indices with respect to $g$ are generated in two arrays.

**See also**

PowerTab, PrimRoot

---

**IntAPTab**

**Function**

Creates a TABle with rows indexed by $a \pmod{m}$ and columns indexed by $b \pmod{n}$. The INTersection of these two Arithmetic Progressions is displayed (if it is nonempty) as a residue class $\pmod{[m,n]}$.

**Syntax**

intaptab

**Commands**

- ↑ Move up
- ↓ Move down
- ← Move left
- → Move right
- a Start at row $a$
- b Start at column $b$
- m Set modulus $m$
- n Set modulus $n$
- p Print (when table is small enough)
- Esc Escape from the environment

**Restrictions**

$m < 10^4$, $n < 10^4$

**Algorithm**

Chinese Remainder Theorem

**See also**

CRT, CRTDem

**Comments**

Reduced residues are written in white, the others in yellow.

*Reference Guide to Turbo Pascal Programs* 79
Jacobi

**Function**
Evaluates the JACOBI symbol \((\frac{P}{Q})\).

**Syntax**
jacobi [P Q]

**Restrictions**
\(|P| < 10^{18}, 0 < Q < 10^{18}\)

**Algorithm**
Modified Euclidean algorithm, using quadratic reciprocity.

**See also**
JacobDem, JacobTab

**Comments**
This program provides a user interface for the function of the same name found in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

JacobDem

**Function**
DEMonstrates the use of quadratic reciprocity to calculate the JACOBi symbol \((\frac{P}{Q})\).

**Syntax**
jacobdem [P Q]

**Restrictions**
\(|P| < 10^{18}, 0 < Q < 10^{18}\)

**Algorithm**
Modified Euclidean algorithm, using quadratic reciprocity.

**See also**
Jacobi, JacobTab

---

JacobTab

**Function**
Generates a TABle of values of the JACOBi function, with 200 values displayed at one time.

**Syntax**
jacobtab

**Commands**
PgUp View the preceding 200 entries
PgDn View the next 200 entries
J Jump to a new position in the table
Q Enter a new denominator Q
P Print 500 lines, starting with the top line displayed
Esc Escape from the environment

**Restrictions**
\(|P| < 10^{18}, 0 < Q < 10^{18}\)

**Algorithm**
Values are calculated as needed, using the function Jacobi.

**See also**
Jacobi, JacobDem

Reference Guide to Turbo Pascal Programs
**LinCon**

**Function**
Finds all solutions of the Linear Congruence \( ax \equiv b \pmod{m} \).

**Syntax**
\[ \text{lincon}[a \ b \ m] \]

**Restrictions**
\[ |a| < 10^{18}, \ |b| < 10^{18}, \ 0 < m < 10^{18} \]

**Algorithm**
The extended Euclidean algorithm is used to find both the number \( g = (a,m) \) and a number \( u \) such that \( au \equiv g \pmod{m} \). If \( g \not| b \) then there is no solution. Otherwise, the solutions are precisely those \( x \) such that \( x \equiv c \pmod{m/g} \) where \( c = ub/g \).

**See also**
LinCnDem

**Comments**
This program provides a user interface for a function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

**LnCnDem**

**Function**
DEMonstrates the method used to find all solutions to the Linear Congruence \( ax \equiv b \pmod{m} \).

**Syntax**
\[ \text{lncndem}[a \ b \ m] \]

**Restrictions**
\[ |a| < 10^{18}, \ |b| < 10^{18}, \ 0 < m < 10^{18} \]

**Algorithm**
See the description given for LinCon.

**See also**
LinCon

---

**LnComTab**

**Function**
Creates a Table of the Linear Combinations \( bx + cy \) of \( b \) and \( c \), with columns indexed by \( x \) and rows indexed by \( y \).

**Syntax**
\[ \text{lncomtab} \]

**Commands**
- ↑: Move up
- ↓: Move down
- ←: Move left
- →: Move right
- x: Left column is \( x \)
- y: Bottom row is \( y \)
- b: Set value of \( b \)
- c: Set value of \( c \)
- Esc: Escape from the environment

*Reference Guide to Turbo Pascal Programs* 81
Lucas

Function
Calculates the LUCAS functions $U_n, V_n \pmod{m}$. Here the $U_n$ are generated by the linear recurrence $U_{n+1} = aU_n + bU_{n-1}$ with the initial conditions $U_0 = 0, U_1 = 1$. The $V_n$ satisfy the same linear recurrence, but with the initial conditions $V_0 = 2, V_1 = a$.

Syntax
`lucas [n [a b] m]` If $n, m$ are specified on the command line, but not $a, b$, then by default $a = b = 1$.

Restrictions
$0 \leq n < 10^{18}, |a| < 10^{18}, |b| \leq 10^{18}, 0 < m \leq 10^{18}$

Algorithm
To calculate $U_n \pmod{m}$, the pair of residue classes $U_{k-1}, U_k \pmod{m}$ is determined for a sequence of values of $k$, starting with $k = 1$. If this pair is known for a certain value of $k$, then it can be found with $k$ replaced by $2k$, by means of the duplication formulae

$$U_{2k-1} = U_k^2 + bU_{k-1}^2,$$
$$U_{2k} = 2bU_{k-1}U_k + aU_k^2. $$

This is called “doubling.” Alternatively, the value of $k$ can be increased by 1 by using the defining recurrence. This is called “sidestepping.” By repeatedly doubling, with sidesteps interspersed as appropriate, eventually $k = n$.

To calculate $V_n \pmod{m}$, the pair $V_k, V_{k+1}$ of residue classes $\pmod{m}$ is determined for a sequence of values of $k$, starting with $k = 0$. The duplication formulae are now

$$V_{2k} = V_k^2 - 2(-b)^k,$$
$$V_{2k+1} = V_kV_{k+1} - a(-b)^k.$$

Instead of sidestepping separately, an arithmetic economy is obtained by doubling with sidestep included by means of the formulae

$$V_{2k+1} = V_kV_{k+1} - a(-b)^k,$$
$$V_{2k+2} = V_k^2 + 2(-b)^{k+1}.$$

By employing these transformations we eventually reach $k = n$.

The $k$ that arise have binary expansions that form initial segments of the binary expansion of $n$, in the same manner as in the alternative powering algorithm discussed in the program PwrDem2.

The system of calculation here is superior to that found in the Fifth
Edition of NZM, where the sidestep formula involves division by 2 and is therefore appropriate only for odd moduli.

**See also** LucasDem, LucasTab, PwrDem2

**Comments**
If \( a = b = 1 \) then \( U_n, V_n \) are the familiar Fibonacci and Lucas sequences \( F_n, L_n \), respectively. This program provides a user interface for the functions LucasU and LucasV found in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

**LucasDem**

**Function**
DEMonstrates the method used to calculate the LUCAS functions \( U_n, V_n \) (mod \( m \)).

**Syntax**
lucasdem \([n \ a \ b \ m]\)

**Restrictions**
\( 0 \leq n < 10^{18}, |a| < 10^{18}, |b| < 10^{18}, 0 < m < 10^{18} \)

**Algorithm**
See the description given for the program Lucas.

**See also** Lucas, LucasDem, PwrDem2

---

**LucasTab**

**Function**
Generates a TABle of values of the LUCAS functions \( U_n, V_n \) (mod \( m \)).

**Syntax**
lucastab

**Commands**
- PgUp Display the preceding 100 values
- PgDn Display the next 100 values
- U Switch from \( V \) to \( U \)
- V Switch from \( U \) to \( V \)
- n Move to a screen with \( n \) on the top line
- a Choose a new value for the parameter \( a \)
- b Choose a new value for the parameter \( b \)
- M Choose a new modulus \( m \)
- P Print the initial 60 rows of the table (\( 0 \leq n \leq 599 \))
- Esc Escape from the environment

**Restrictions**
\( 0 \leq n < 10^6, |a| < 10^6, |b| < 10^6, 0 < m < 10^6 \)

**See also** Lucas, LucasDem

---

**Mult**

**Function**
MULTiplies residue classes. If \( a, b, \) and \( m \) are given with \( m > 0 \), then \( c \) is found so that \( c \equiv ab \) (mod \( m \)) and \( 0 \leq c < m \).
Syntax  mult [a b m]
Restrictions  |a| < 10^{18}, |b| < 10^{18}, 0 < m < 10^{18}
Algorithm  If \( m \leq 10^9 \) then \( ab \) is reduced modulo \( m \). If \( 10^9 < m \leq 10^{12} \) then we write \( a = a_110^9 + a_0 \), and compute \( a_1b10^9 + a_0b \) modulo \( m \), with reductions modulo \( m \) after each multiplication. Thus all numbers encountered have absolute value at most \( 10^{18} \). If \( 10^{12} < m < 10^{18} \) then we write \( a = a_110^9 + a_0 \), \( b = b_110^9 + b_0 \); we compute \( ab/m \) in floating-point real arithmetic and let \( q \) be the integer nearest this quantity; we write \( q = q_110^9 + q_0 \); \( m = m_110^9 + m_0 \). Then

\[
ab - qm = \left((a_1b_1 - q_1m_1)10^9 + a_1b_0 + a_0b_1 - q_1m_0 - q_0m_1\right)10^9 + a_0b_0 - q_0m_0.
\]

The right hand side can be reliably evaluated, and this quantity has absolute value less than \( m \). If it is negative we add \( m \) to it to obtain the final result. The assumption is that the machine will perform integer arithmetic accurately for integers up to \( 4 \cdot 10^{18} \) in size. The object is to perform congruence arithmetic with a modulus up to \( 10^{18} \) without introducing a full multiprecision package.

See also  MultDem1, MultDem2, MultDem3
Comments  This program provides a user interface for the function of the same name found in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

**MultDem1**

Function  DEMonstrates the method employed by the program MULT when \( 10^9 < m < 10^{12} \).
Syntax  multdem1
Restrictions  |a| < 10^{18}, |b| < 10^{18}, 0 < m < 10^{18}
Algorithm  See Problem *21, Section 2.4, p. 83, of the Fifth Edition of NZM.
See also  Mult, MultDem2, MultDem3

**MultDem2**

Function  DEMonstrates the method used by the program MULT when \( 10^{12} < m < 10^{18} \).
Syntax  multdem2
Restrictions  |a| < 10^{18}, |b| < 10^{18}, 0 < m < 10^{18}
### MultDem3

**Function**: DEMonstrates the method used by the program MULT, in which the methods of MultDem1 and MultDem2 are merged.

**Syntax**: `multdem3`

**Restrictions**: $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

**Algorithm**: See the description given for the program Mult.

**See also**: Mult, MultDem1, MultDem2

### Order

**Function**: Calculates the ORDER of a reduced residue class $a \pmod{m}$. That is, it finds the least positive integer $h$ such that $a^h \equiv 1 \pmod{m}$.

**Syntax**: `order [a m [c]]`

**Restrictions**: $|a| < 10^{18}$, $0 < m < 10^{18}$, $0 < c < 10^{18}$

**Algorithm**: The parameter $c$ should be any known positive number such that $a^c \equiv 1 \pmod{m}$. For example, if $m$ is prime then one may take $c = m - 1$. If a value of $c$ is not provided by the user, or if the value provided is incorrect, then the program assigns $c = \text{Carmichael}(m)$. (This involves factoring $m$ by trial division.) Once $c$ is determined, then $c$ is factored by trial division. Prime divisors of $c$ are removed, one at a time, to locate the smallest divisor $d$ of $c$ for which $a^d \equiv 1 \pmod{m}$. This number is the order of $a$ modulo $m$.

**See also**: OrderDem

**Comments**: This program provides a user interface for a function of the same name found in the unit NoThy. To see how the algorithm is implemented, inspect the file `nothy.pas`.

### OrderDem

**Function**: DEMonstrates the method used to calculate the order of a reduced residue class $a \pmod{m}$.

**Syntax**: `order [a m [c]]`

*Reference Guide to Turbo Pascal Programs* 85
Restrictions $|a| < 10^{18}$, $0 < m < 10^{18}$, $0 < c < 10^{18}$

Algorithm See the description given for the program Order.

See also Order

---

**P–1**

**Function** Factors a number $n$ using the Pollard $p – 1$ method.

**Syntax** `p-1 [n [a]]` If $n$ is specified on the command line, but not $a$, then by default $a = 2$.

**Restrictions** $1 < n < 10^{18}$, $1 < a < 10^{18}$

**Algorithm** The powering algorithm is used to calculate $a^k \pmod{n}$ for increasingly large $k$, in the hope that a $k$ will be found such that $1 < (a^k-1, n) < n$. This method is generally fast for those $n$ with a prime factor $p$ such that $p – 1$ is composed only of small primes.

See also P-1Dem, Rho, RhoDem, Factor

---

**P–1Dem**

**Function** Demonstrates the method used by the Pollard $p – 1$ factoring scheme.

**Syntax** `p-1dem`

**Restrictions** $1 < n < 10^{18}$, $1 < a < 10^{18}$

**Algorithm** See the description given for the program P-1.

See also P-1

---

**PascalsT**

**Function** Constructs a table of PASCAL’S Triangle $\binom{n}{k} \pmod{m}$. Rows are indexed by $n$, columns by $k$. Up to 20 rows and 18 columns are displayed at one time.

**Syntax** `pascalst`

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Display the preceding 20 rows</td>
</tr>
<tr>
<td>↓</td>
<td>Display the next 20 rows</td>
</tr>
<tr>
<td>←</td>
<td>Display the preceding 20 columns</td>
</tr>
<tr>
<td>→</td>
<td>Display the next 20 columns</td>
</tr>
<tr>
<td>T</td>
<td>Move to the top of the triangle</td>
</tr>
<tr>
<td>M</td>
<td>Choose a new modulus</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

Reference Guide to Turbo Pascal Programs
**Restrictions**

0 ≤ k ≤ n < 10^4, 0 < m < 10^3

**Algorithm**

The rows are calculated inductively by the recurrence \((\binom{n-1}{k-1}) + \binom{n}{k} = \binom{n+1}{k}\). The entire nth row is calculated, where n is the top row on the current screen. Other entries in the screen are calculated from the top row.

---

**Phi**

**Function**

Calculates the Euler PHI function of n.

**Syntax**

```pascal
phi [n]
```

**Restrictions**

1 ≤ n < 10^18

**Algorithm**

The canonical factorization of n is found by trial division, and then \(\phi(n)\) is found by means of the formula \(\phi(n) = \prod_{p\mid n} p^{a-1}(p-1)\).

**Comments**

This program provides a user interface for a function of the same name found in the unit NoThy. To see how the algorithm is implemented, inspect the file `nothy.pas`.

---

**Pi**

**Function**

Determines the number \(\pi(x)\) of primes not exceeding an integer \(x\).

**Syntax**

```pascal
pi [x]
```

**Restrictions**

2 ≤ x \(10^6\)

**Algorithm**

Primes up to 31607 are constructed, by sieving. These primes are used as trial divisors, to sieve intervals of length \(10^4\) until \(x\) is reached.

**Comments**

This program would run perfectly well up to \(10^9\), but as the the running time is roughly linear in \(x\), the smaller limit is imposed to avoid excessive running times. For faster methods of computing \(\pi(x)\), see the following papers.


---

*Reference Guide to Turbo Pascal Programs* 87
**PolySolv**

**Function**
Finds all solutions of a given polynomial congruence \( P(x) \equiv 0 \pmod{m} \).

**Syntax**
```
polySolv
```

**Commands**
<table>
<thead>
<tr>
<th>C</th>
<th>Count the zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Define the polynomial</td>
</tr>
<tr>
<td>M</td>
<td>Choose the modulus</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

**Restrictions**
\( 1 \leq m < 10^4 \), \( P(x) \) must be the sum of at most 20 monomials, only the first 100 zeros found are displayed on the screen.

**Algorithm**
The polynomial is evaluated at every residue class modulo \( m \).

**See also**
SqrtModP

**Comments**
The running time here is roughly linear in \( m \). When \( m \) is large there is a much faster way. By the Chinese Remainder Theorem it is enough to consider primepower values of \( m \). By Hensel’s lemma, this in turn can be reduced to the consideration of prime moduli. In the case of a prime modulus \( p \), the roots of \( P(x) \) modulo \( p \) can be found by calculating \((P(x),(x-a)^{(p-1)/2}−1)\) for various values of \( a \). Here the gcd being calculated is that of two polynomials defined mod \( p \). In the first step of the Euclidian algorithm, the remainder when \((x−a)^{(p−1)/2}−1\) is divided by \( P(x) \) should be calculated by applying the powering algorithm to determine \((x−a)^{(p−1)/2} \pmod{p, P(x)}\). This approach extends to provide an efficient method of determining the factorization of \( P(x) \pmod{p} \). For more information, see David G. Cantor and Hans Zassenhaus, A new algorithm for factoring polynomials over finite fields, Math. Comp. 36 (1981), 587–592.

**Power**

**Function**
Computes \( a^k \pmod{m} \) in the sense that it returns a number \( c \) such that \( 0 \leq c < m \) and \( c \equiv a^k \pmod{m} \).

**Syntax**
```
power [a k m]
```

**Restrictions**
\( |a| < 10^{18}, 0 \leq k < 10^{18}, 0 < m < 10^{18} \)

**Algorithm**
Write \( k \) in binary, say \( k = \sum_{j \in \mathbb{Z}} 2^j \). The numbers \( a^{2^j} \pmod{m} \) are constructed by repeated squaring; whenever a \( j \in \mathbb{Z} \) is encountered, the existing product is multiplied by the factor \( a^{2^j} \).

**See also**
PowerTab, PwrDem1a, PwrDem1b, PwrDem2

Reference Guide to Turbo Pascal Programs
Comments  This program provides a user interface for a function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

PowerTab

Function  Constructs a TABle of POWERs $a^k \pmod{m}$. Up to 100 powers are displayed at a time.

Syntax  

Commands  

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgUp</td>
<td>Display the preceding 10 rows</td>
</tr>
<tr>
<td>PgDn</td>
<td>Display the next 10 rows</td>
</tr>
<tr>
<td>B</td>
<td>Change the base</td>
</tr>
<tr>
<td>E</td>
<td>Move to a new exponent</td>
</tr>
<tr>
<td>M</td>
<td>Change the modulus</td>
</tr>
<tr>
<td>P</td>
<td>Print the first 60 lines of the table</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

Restrictions  $|a| < 10^6$, $0 \leq k < 10^6$, $0 < m < 10^6$

Algorithm  The first entry on the screen is computed by the powering algorithm. Then the remaining entries on the screen are determined inductively.

See also  CngArTab, Power, PwrDem1a, PwrDem1b, PwrDem2

PrimRoot

Function  Finds the least primitive root $g$ of a prime number $p$, such that $g > a$.

Syntax  

primroot [p [a]] If $p$ is specified on the command line but not $a$, then by default $a = 0$.

Restrictions  $2 \leq p < 10^{18}$, $|a| < 10^{18}$

Algorithm  The prime factors $q_1, q_2, \ldots, q_r$ of $p - 1$ are found by trial division. Then $g$ is a primitive root of $p$ if and only if both $g^{p-1} \equiv 1 \pmod{p}$ and $g^{(p-1)/q_i} \not\equiv 1 \pmod{p}$ for all $i$, $1 \leq i \leq r$. When a $g$ is found that satisfies these conditions, not only is $g$ a primitive root of $p$, but also the primality of $p$ is rigorously established. The algorithm employed by the program ProveP proceeds along these lines, but with some short cuts.

See also  Order, OrderDem, ProveP

Comments  This program provides a user interface for a function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

Reference Guide to Turbo Pascal Programs  89
**ProveP**

<table>
<thead>
<tr>
<th>Function</th>
<th>PROVEs that a given number ( p ) is Prime.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>provep [p]</code></td>
</tr>
<tr>
<td>Restrictions</td>
<td>( 2 \leq p &lt; 10^{18} )</td>
</tr>
</tbody>
</table>
| Algorithm | Trial division is applied to \( p - 1 \). Whenever a prime factor \( q \) of \( p - 1 \) is found, say \( q^k \| (p-1) \), attempts are made to find an \( a \) such that \( a^{p-1} \equiv 1 \pmod{p} \) but \( (a^{(p-1)/q} - 1, p) = 1 \). Suppose that such an \( a \) is found, and that \( p' \| p \). Let \( d \) denote the order of \( a \) modulo \( p' \). Then \( d \| (p-1) \) but \( d \not| (p-1)/q \), and hence \( q^k \| d \). But by Fermat’s congruence \( d | (p'-1) \), and hence it can be asserted that \( q^k \| (p'-1) \) for every prime factor \( p' \) of \( p \). In other words, all prime factors \( p' \) of \( p \) are \( \equiv 1 \pmod{q^k} \). If, for a given \( q \), 200 unsuccessful attempts are made to find an admissible \( a \), then presumably \( p \) is composite, and the program quits. Otherwise, the numbers \( q^k \) found are multiplied together to form a product \( s \). Every prime factor \( p' \) of \( p \) is \( \equiv 1 \pmod{s} \). If \( s > \sqrt{p} \) then there can be at most one such prime, and the proof is complete. If \( p^{1/3} < s \leq p^{1/2} \) then there can be at most two such primes, say \( p = p_1 p_2 \). Write \( p_1 \) in base \( s \), \( p_1 = r_1 s + 1 \). Then \( p = r_1 r_2 s^2 + (r_1 + r_2) s + 1 \), and the coefficients of this polynomial in \( s \) can be found by expanding \( p \) in base \( s \), say \( p = c_2 s^2 + c_1 s + 1 \). Then \( r_1 \) and \( r_2 \) are roots of the quadratic equation \( (x-r_1)(x-r_2) = x^2 - c_1 x + c_2 \), and hence the discriminant \( c_1^2 - 4c_2 \) must be a perfect square. In the unlikely event that this quantity is a perfect square, we are led to a factorization of \( p \); otherwise we have a proof that \( p \) is prime.

If a point is reached at which it would take less time to test \( p \) for divisibility by numbers \( d \equiv 1 \pmod{s} \), \( d \leq \sqrt{p} \) than has already been spent trying to factor \( p-1 \), then the program automatically switches to this latter approach.

The trial division of \( p-1 \) can be interrupted by touching a key, and the user can then supply a prime factor \( q \) of the remaining unfactored portion. The user is responsible for verifying that \( q \) is prime.

By this method we see that proving the primality of \( p \) is no harder than factoring \( p-1 \), and that for many \( p \) it is easier. Further methods of proving primality have been developed that are faster than the best known factoring methods. The mathematics exploited by these methods is much more sophisticated. For more precise information, consult the following papers.


Reference Guide to Turbo Pascal Programs
**PwrDem1a**

**Function**
DEMonstrates the powering algorithm.

**Syntax**
pwrdem1a \[a \ k \ m\]

**Restrictions**
\[|a| < 10^{18}, \ 0 \leq k < 10^{18}, \ 0 < m < 10^{18}\]

**Algorithm**
See the description given for the program Power.

**See also**
Power, PwrDem1b, PwrDem2

---

**PwrDem1b**

**Function**
An alternative DEMonstration of the powering algorithm.

**Syntax**
pwrdem1b \[a \ k \ m\]

**Restrictions**
\[|a| < 10^{18}, \ 0 \leq k < 10^{18}, \ 0 < m < 10^{18}\]

**Algorithm**
See the description given for the program Power.

**See also**
Power, PwrDem1a, PwrDem2

---

**PwrDem2**

**Function**
DEMonstrates an alternative powering algorithm.

**Syntax**
pwrdem2 \[a \ k \ m\]

**Restrictions**
\[|a| < 10^{18}, \ 0 \leq k < 10^{18}, \ 0 < m < 10^{18}\]

**Algorithm**
A sequence of powers of \(a\) is generated, in which the binary expansions of the exponents form initial segments of the binary expansion of \(k\). For example, if \(k = 10111\) in binary, then (with all exponents written in binary) we start with \(a^1\), square to form \(a^{10}\), square again to form \(a^{100}\), multiply by \(a\) to form \(a^{101}\), square this to form \(a^{1010}\), multiply by \(a\) to form \(a^{1011}\), square this to form \(a^{10110}\), and finally multiply by \(a\) to form \(a^{10111}\). Of course all multiplications are carried out modulo \(m\). In the original method used by the program Power, the binary expansions of the exponents form terminal segments of the binary expansion of \(k\). The number of multiplications is exactly the same in the two methods, but this alternative method has an advantage in situations in which multiplication by \(a\) is fast for some reason. For example, in powering a matrix \(A\), multiplication by \(A\) is fast if \(A\) is sparse. Similarly, in computing \(P(x)^k\), multiplication by \(P(x)\) is fast if \(P(x)\) has few monomial terms. The repeated doubling from the top down seen here is also appropriate to the calculation of solutions of linear recurrences.

Reference Guide to Turbo Pascal Programs 91
See also  Power, PwrDem1a, PwrDem1b, LucasDem

QFormTab

**Function**
Generates a TABle of all reduced binary Quadratic FORMs \( f(x, y) = ax^2 + bxy + cy^2 \) of given discriminant. These forms are reduced only in the sense defined in §3.5 of NZM. Hence if \( d > 0 \) then the reduced forms are not necessarily inequivalent. For each form, the content \( (a, b, c) \) is calculated.

**Syntax**
\[ \text{qformtab} \]

**Commands**
- \( \text{PgUp} \) Display the preceding 20 rows
- \( \text{PgDn} \) Display the next 20 rows
- \( d \) Choose a new discriminant
- \( P \) Print the first 600 lines of the table
- \( \text{Esc} \) Escape from the environment

**Restrictions**
\[ |d| < 10^6, \text{at most 5000 forms are displayed} \]

**Algorithm**
Detailed search for all triples satisfying the definition. Thus the running time is essentially linear in \( |d| \). This program could run for \( |d| \) up to \( 10^9 \), but the stricter limit is imposed to avoid excessive running times. For faster methods, see the discussion of the program ClaNoTab.

See also  ClaNoTab, Reduce

Rat

**Function**
Finds the RATional number \( a/q \) with least \( q \) such that the initial decimal digits of \( a/q \) coincide with those of a given real number \( x \).

**Syntax**
\[ \text{rat } [x] \]

**Restrictions**
\[ |a| \leq 10^{18}, \ 1 \leq q \leq 10^{18} \]

**Algorithm**
Suppose that \( k \) decimal digits of \( x \) are given after the decimal point. Put \( \delta = 0.5 \cdot 10^{-k} \). We want to find \( a/q \) with \( q \) minimal such that \( |x - a/q| \leq \delta \). By the continued fraction algorithm the least \( i \) is found such that \( |x - h_i/k_i| \leq \delta \). Then the desired rational number is given by \( a = ch_{i-1} + h_{i-2}, \ q = ck_{i-1} + k_{i-2} \) where \( c \) is the least positive integer such that \( a/q \) lies in the specified interval. Since this inequality holds when \( c = a_i \), it suffices to search the interval \([1, a_i]\).

Reduce

**Function**
REDUCEs a binary quadratic form \( f(x, y) = ax^2 + bxy + cy^2 \). If the three coefficients are given on the command line, then a reduced form \( g(x, y) \)
is found, with \( g \) equivalent to \( f \). The discriminant \( d \) of these forms is also reported. A proper representation of \( a \) by \( g \) is also noted, and then the program terminates. If the coefficients are not given on the command line, then an environment for manipulating forms is entered. When a form is being reduced in this environment, a chain of equivalences is displayed, along with the matrix \( M \) that gives the equivalence, and the operation \( S \) or \( T^n \) that was applied to derive the new form from that in the preceding row of the table. Here \( S = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \) and \( T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \).

The user also has the option of applying the operations \( S, T, \) and \( T^{-1} \), one at a time. The table will hold up to 500 forms.

In the case that \( d > 0 \), the form is reduced only to the extent that \( |a| < b \leq |a| < |a| \) or \( 0 \leq b \leq |a| = |c| \), and consequently two reduced forms may be equivalent.

**Syntax**  
reduce \([a \ b \ c]\)

**Restrictions**  
\(|a| < 10^{18}, \ |b| < 10^{18}, \ |c| < 10^{18}\)

**Commands**  
- PgUp: Display the preceding 6 rows
- PgDn: Display the next 6 rows
- a: Enter a new coefficient \(a\)
- b: Enter a new coefficient \(b\)
- c: Enter a new coefficient \(c\)
- R: Reduce the form at the bottom of the table
- S: Apply the transformation \(S\)
- T: Apply the transformation \(T\)
- I: Apply the transformation \(T^{-1}\)
- M: Toggle between displaying \(M: g \rightarrow f\) and \(M: f \rightarrow g\)
- P: Print the table
- Esc: Escape from the environment

**See also**  
ClaNoTab, QFormTab

---

**Rho**

**Function**  
Factors a given composite integer \(n\) by using Pollard’s RHO method. This program should only be applied to numbers that are already known to be composite; if it is applied to a prime number then it will run endlessly without reaching any conclusion. The program can be interrupted by touching any key on the keyboard.

**Syntax**  
rho \([n \ [c]]\) If \(n\) is specified on the command line but not \(c\), then \(c = 1\) by default.

**Restrictions**  
\(1 < n < 10^{18}, \ |c| < 10^{18}\)

Reference Guide to Turbo Pascal Programs 93
Algorithm
Let \( u_0 = 0 \), and for \( i \geq 0 \) let \( u_{i+1} = u_i^2 + c \). The \( u_i \) are calculated modulo \( n \), and for each \( i \) the quantity \( (u_{2i} - u_i, n) \) is determined, in the hope of finding a proper divisor of \( n \). The numbers \( u_i \) are not stored: At any one time only \( u_i \) and \( u_{2i} \) are known. If a proper divisor is found, it is not necessarily prime, and if it is prime it is not necessarily the least prime divisor of \( n \). Various values of \( c \) may be used, but \( c = 0 \) and \( c = -2 \) should be avoided.

See also RhoDem, P-1, P-1Dem, Factor

---

**RhoDem**

**Function**
DEMonstrates the Pollard RHO factoring scheme.

**Syntax**
rhodem [n]

**Restrictions**
1 < \( n \) < 10^{18}, \ |c| < 10^{18}

**Algorithm**
See description given for the program Rho.

**See also**
Rho, P-1, P-1Dem, Fac

---

**RSA**

**Function**
Provides an environment for encrypting messages by means of the RSA method. The encrypting history is displayed.

**Syntax**
rsa

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Set the size of the blocks</td>
</tr>
<tr>
<td>E</td>
<td>Encode</td>
</tr>
<tr>
<td>P</td>
<td>Print the data</td>
</tr>
<tr>
<td>R</td>
<td>Enter a message as a sequence of residue classes</td>
</tr>
<tr>
<td>T</td>
<td>Enter a message in text form</td>
</tr>
<tr>
<td>V</td>
<td>Choose variables: modulus ( m ), exponent ( k ), etc.</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

**Restrictions**
The block size must lie between 1 and 17, the text must consist of at most 80 characters, \( 0 < k < m < 10^{18} \)

**Algorithm**
Each residue class \( a \) \((mod \ m)\) is replaced by \( b \equiv a^k \) \((mod \ m)\). To decode, replace \( b \) by \( b^{k'} \) \((mod \ m)\) where \( 0 < k' < m \) and \( kk' \equiv 1 \) \((mod \ \phi(m))\).

---

**SimLinDE**

**Function**
Gives a complete parametric representation of the solutions to a system of SIMultaneous LINear Diophantine Equations \( Ax = b \). The user may request that the calculations be displayed.

Reference Guide to Turbo Pascal Programs
Syntax: simlinde

Restrictions: A is $m \times n$ where $1 \le m \le 10$, $1 \le n \le 10$, all numbers occurring must have absolute value not exceeding $10^{18}$

Algorithm: Row operations and changes of variable are performed until the system is in diagonal form. The full Smith normal form is not reached. This method is prone to overflow. The program as written makes no special effort to avoid overflow, but reports when it has occurred.

---

**SlowGCD**

Function: Times the calculation of the greatest common divisor of two numbers $b$ and $c$, when only the definition is used. The only purpose in this is to provide a comparison with FastGCD.

Syntax: slowgcd

Restrictions: $1 \le b < 10^9$, $1 \le c < 10^9$

Algorithm: For each $d$, $1 \le d \le \min(|b|, |c|)$, trial divisions are made to determine whether $d|b$ and $d|c$. A record is kept of the largest such $d$ found. Since the running time is essentially linear in $\min(|b|, |c|)$, only small arguments should be used.

See also: FastGCD, GCD

---

**SPsP**

Function: Executes the Strong PseudoPrime test base $a$ to the number $m$. This provides a rigorous proof of compositeness. If $m$ survives such a test then it is not necessarily prime, but it is called a “probable prime” because pseudoprimes (i.e., composite probable primes) seem to form a sparse set.

Syntax: spsp $[a]$ $m$ If $m$ is specified on the command line, but not $a$, then by default $a = 2$.

Restrictions: $|a| < 10^{18}$, $2 < m < 10^{18}$

Algorithm: The strong pseudoprime test, as invented by John Selfridge and others. For a full description see NZM, p. 78.

See also: SPsPDem, ProveP

---

**SPsPDem**

Function: DEMonstrates the Strong PSeudoPrime test.

Reference Guide to Turbo Pascal Programs
Syntax  \[ \text{spsp} \ [a\ m] \] If \( m \) is specified on the command line, but not \( a \), then by default \( a = 2 \).

Restrictions  \(|a| < 10^{18}, 2 < m < 10^{18}\)

See also  SPsP, ProveP

---

**SqrtDem**

**Function**  DEMonstrates the calculation executed by the program SqrtModP.

**Syntax**  \[ \text{sqrtdem} \ [a\ p] \]

**Restrictions**  \(|a| < 10^{18}, 2 \leq p < 10^{18}\)

**Algorithm**  See the description given for the program SqrtModP

See also  SqrtModP

---

**SqrtModP**

**Function**  Calculates the SQuareRooT Modulo a given Prime number \( p \). If the congruence \( x^2 \equiv a \pmod{p} \) has a solution, then the unique solution \( x \) such that \( 0 \leq x \leq p/2 \) is returned.

**Syntax**  \[ \text{srtmodp} \ [a\ p] \]

**Restrictions**  \(|a| \leq 10^{18}, 2 \leq p \leq 10^{18}\)


See also  SqrtDem

Comments  This program provides a user interface for a function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file nothy.pas.

---

**SumsPwrs**

**Function**  Finds all representations of \( n \) as a sum of \( s \) \( k \)-th powers, and counts them in various ways.

**Syntax**  \[ \text{sumspwrs} \ [n\ s\ k] \]

**Restrictions**  \( 1 \leq n < 10^{11}, 2 \leq s \leq 75, 2 \leq k \leq 10 \)

---

Reference Guide to Turbo Pascal Programs
Algorithm  After \( s - 1 \) summands have been chosen, a test is made as to whether the remainder is a \( k \)-th power. Summands are kept in monotonic order; the multiplicity is recovered by computing the appropriate multinomial coefficient. In some cases, such as sums of two squares, much faster methods exist for finding all representations.

See also  Wrg1Tab, Wrg2Tab, WrgStTab, WrgCnTab

### Wrg1Tab

**Function**  Creates a TABle of the number \( r(n) \) of representations of \( n = \sum_{i=1}^{s} x_i^k \) as a sum of \( s \) \( k \)-th powers, as in WARing’s problem. If \( k > 2 \) then the \( x_i \) are non-negative, but for \( k = 2 \) the \( x_i \) are arbitrary integers.

**Syntax**  \texttt{wrg1tab}

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgUp</td>
<td>Move up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Move down</td>
</tr>
<tr>
<td>s</td>
<td>Set ( s ), the number of summands</td>
</tr>
<tr>
<td>k</td>
<td>Set ( k ), the exponent</td>
</tr>
<tr>
<td>N</td>
<td>Start the table at 10n</td>
</tr>
<tr>
<td>p</td>
<td>Print the table</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

**Restrictions**  \( 1 \leq s \leq 75, 2 \leq k \leq 10, 1 \leq n \leq 10^{11} \)

**Algorithm**  Search for representations, with summands in monotonic order. The multiplicity of a representation is recovered by multiplying by the appropriate multinomial coefficient.

**See also**  SumsPwrs, Wrg2Tab, WrgStTab, WrgCnTab

### Wrg2Tab

**Function**  Creates a TABle of the least number \( s \) of \( k \)-th powers required to represent \( n \), in connection with WARing’s problem.

**Syntax**  \texttt{wrg2tab}

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgUp</td>
<td>Move up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Move down</td>
</tr>
<tr>
<td>k</td>
<td>Set ( k ), the exponent</td>
</tr>
<tr>
<td>N</td>
<td>Start the table at 10n</td>
</tr>
<tr>
<td>p</td>
<td>Print the table</td>
</tr>
<tr>
<td>Esc</td>
<td>Escape from the environment</td>
</tr>
</tbody>
</table>

**Restrictions**  \( 2 \leq k \leq 10, 1 \leq n \leq 10^4 \)
Algorithm
For $s \leq k$, the numbers represented are found by allowing a $k$-tuple of variables run over all possible values, with coordinates in monotonic order. For $s > k$, all possible $k$-th powers are added to numbers already represented, until more than half the numbers have been represented. Then all possible $k$-th powers are subtracted from numbers not represented.

See also
SumsPwrs, Wrg1Tab, Wrg2Tab, WrgCnTab

WrgCnTab

Function
Creates a TABLE of the number of solutions of the congruence $\sum_{i=1}^{s} x_i^k \equiv n \pmod{m}$, in connection with WARRing’s problem.

Syntax
wrgcntab

Commands
- PgUp: Move up
- PgDn: Move down
- n: First line displayed is $n$
- m: Set the modulus $m$
- p: Print the table
- Esc: Escape from the environment

Restrictions
$1 \leq s \leq 75, 2 \leq k \leq 10, 1 \leq m < 5000$

Algorithm
First a list of all $k$-th power residues $r$ is constructed, with the number of solutions of $x^k \equiv r \pmod{m}$ is recorded. Summands run over monotonically ordered residues. To recover the multiplicity of a representation, one must multiply by the appropriate multinomial coefficient and by the multiplicities of the summands.

See also
SumsPwrs, Wrg1Tab, Wrg2Tab, WrgStTab
A collection of basic routines are provided for use in more advanced programs. These routines are accessed in one of two ways. First, there are files with the extension .i that may be included in another program. For example, to measure the running time of a program you may type `{$I timer.i }`. (The space after the .i is essential here.) The effect will be the same as if the text of the file timer.i had been pasted into your program at this point. Second, a library of 17 number-theoretic routines is provided in the Turbo Pascal unit nothy.tpu. This is a compiled module that the compiler will use when your program is compiled. The source code for this unit is in the file nothy.pas. To invoke this unit, the initial lines of your program should include commands of the following sort:

```pascal
program TwoSquares; {Use the method of Problem 6 on p. 333 to write a prime p as a sum of two squares}
{$N+,E+}
uses nothy;
```

Most of the routines in NoThy accept integers as variables of type comp, with a size up to $10^{18}$. This type is available only after the compiler directive `{$N+}` has been given. Such variables are calculated on the arithmetic coprocessor, in floating point. If no coprocessor is found, then the program will crash, unless the compiler directive `{$E+}` has also been given, in which case the numerical work of the coprocessor will be emulated in software.

### Canonic procedure

<table>
<thead>
<tr>
<th>Function</th>
<th>Calculates the canonical factorization of an integer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td><code>canonic(n: comp; var k: integer; var p: primes; var m: multiplicity; var Prog: Boolean)</code></td>
</tr>
<tr>
<td>Remarks</td>
<td>This procedure uses two variable types defined within the NoThy unit: primes = array[1..15] of comp; multiplicity = array[1..15] of integer. k is the number of distinct primes dividing n; these primes are stored, in increasing order, in the array p. The multiplicity to which these primes divide n is recorded in the corresponding location in the array m. If Prog</td>
</tr>
</tbody>
</table>
= True then the progress in computing the factorization is reported to the screen. Since the underlying method is trial division, performance will be slow whenever \( n \) has a very large prime factor. In such a case, execution may be interrupted by typing any key.

Restrictions \( 1 \leq n \leq 10^{18} \)

<table>
<thead>
<tr>
<th>Carmichael function</th>
<th>NoThy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Computes the Carmichael function of ( n ). That is, the least positive integer ( c ) such that ( a^c \equiv 1 \pmod{n} ) whenever ((a,n) = 1).</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>\texttt{carmichael}(n: comp)</td>
</tr>
<tr>
<td><strong>Result type</strong></td>
<td>comp</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>Since ( n ) is factored by trial division, performance will be slow if ( n ) has a very large prime factor. In such a case, the execution may be interrupted by typing any key.</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>( 1 \leq n \leq 10^{18} )</td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td>Phi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition function</th>
<th>NoThy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Given ( a ) and ( m ), the number ( b ) is returned where ( b \equiv a \pmod{m} ) and ( 0 \leq b &lt; m ).</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>\texttt{condition}(a, m: comp)</td>
</tr>
<tr>
<td><strong>Result type</strong></td>
<td>comp</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>(</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CRThm procedure</th>
<th>NoThy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Determines the intersection of two given arithmetic progressions.</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>\texttt{CRThm}(a1, m1, a2, m2: comp; var a, m: comp)</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>If the intersection is empty then the value ( m = 0 ) is returned.</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>(</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DetModM function</th>
<th>det.i</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Calculates the determinant of an ( n \times n ) integral matrix ( A = [a_{ij}] ) modulo ( m ).</td>
</tr>
</tbody>
</table>

100 Turbo Pascal Programming Resources
<table>
<thead>
<tr>
<th>Declaration</th>
<th>det(A: matrix; n: integer; m: comp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result type</td>
<td>comp</td>
</tr>
<tr>
<td>Remarks</td>
<td>Before this function is called, the following variable type must be defined: matrix = array[1..9] of array[1..9] of comp.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>$</td>
</tr>
</tbody>
</table>

### GCD function

<table>
<thead>
<tr>
<th>Declaration</th>
<th>gcd(b, c: comp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result type</td>
<td>comp</td>
</tr>
<tr>
<td>Remarks</td>
<td>The gcd is undefined when $b = c = 0$.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>$</td>
</tr>
</tbody>
</table>

### GetInput function

<table>
<thead>
<tr>
<th>Declaration</th>
<th>getinput(x, y: integer; prompt, comm: string; a, b: comp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result type</td>
<td>comp</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function may be modified for more specialized tasks, as is the case with the function GetDisc found in the program QFormTab. Any program using this function must declare the unit CRT in the uses statement.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>$1 \leq x \leq 80$, $1 \leq y \leq 25$, $</td>
</tr>
<tr>
<td>Examples</td>
<td>See the files factor.pas, phi.pas.</td>
</tr>
</tbody>
</table>

### GetNextP function

<table>
<thead>
<tr>
<th>Declaration</th>
<th>getnextp(x: longint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result type</td>
<td>longint</td>
</tr>
<tr>
<td>Remarks</td>
<td>If $x &lt; 0$ or $x &gt; 10^9$ then the value 0 is returned.</td>
</tr>
</tbody>
</table>
Restrictions \(1 \leq x \leq 10^9\)

### Jacobi function

**Function** Calculates the Jacobi symbol \(\left( \frac{p}{q} \right)\).

**Declaration** `jacobi(p, q: comp)`

**Result type** `integer`

**Restrictions** \(|P| \leq 10^{18}, 1 \leq Q \leq 10^{18}, Q \text{ odd.}\)

### LinCon procedure

**Function** Solves the linear congruence \(a_1 x \equiv a_0 \pmod{m}\). If solutions exist then they form an arithmetic progression, \(x \equiv a \pmod{m_1}\).

**Declaration** `lincon(a1, a0, m: comp; var a, m1: comp)`

**Remarks** If \((a_1, m)\) \(\not| a_0\) then the congruence has no solution, and the values \(a = (a_1, m), m_1 = 0\) are returned.

**Restrictions** \(|a_i| \leq 10^{18}, 1 \leq m \leq 10^{18}\)

### LucasU function

**Function** Computes \(U_n \pmod{m}\). Here \(U_n\) is the Lucas sequence with parameters \(a\) and \(b\), defined by the recurrence \(U_{n+1} = aU_n + bU_{n-1}\), with initial conditions \(U_0 = 0, U_1 = 1\). If \(a = b = 1\) then these are the Fibonacci numbers \(F_n\).

**Declaration** `lucasu(n, a, b, m: comp)`

**Result type** `comp`

**Restrictions** \(0 \leq n \leq 10^{18}, |a| \leq 10^{18}, |b| \leq 10^{18}, 1 \leq m \leq 10^{18}\)

**See also** LucasV

### LucasV function

**Function** Computes \(V_n \pmod{m}\). Here \(V_n\) is the Lucas sequence with parameters \(a\) and \(b\), defined by the recurrence \(V_{n+1} = aV_n + bV_{n-1}\), with initial conditions \(V_0 = 0, V_1 = 1\). If \(a = b = 1\) then these are the Lucas numbers \(L_n\).

**Declaration** `lucasv(n, a, b, m: comp)`

102  
*Turbo Pascal Programming Resources*
**Result type**  comp

**Restrictions**  $0 \leq n \leq 10^{18}$, $|a| \leq 10^{18}$, $|b| \leq 10^{18}$, $1 \leq m \leq 10^{18}$

**See also**  LucasU

---

**Mult function**  NoThy

**Function**  Given $a$, $b$, and $m$, returns the number $c$ such that $c \equiv ab \pmod{m}$ and $0 \leq c < m$.

**Declaration**  mult(a, b, m: comp)

**Result type**  comp

**Remarks**  This allows congruence arithmetic for $m$ up to $10^{18}$ without need for multiple precision arithmetic.

**Restrictions**  $|a| \leq 10^{18}$, $|b| \leq 10^{18}$, $1 \leq m \leq 10^{18}$

---

**Order function**  NoThy

**Function**  Given $a$, $m$, and $c$ such that $a^c \equiv 1 \pmod{m}$, the least positive integer $h$ such that $a^h \equiv 1 \pmod{m}$ is returned.

**Declaration**  order(a, m, c: comp)

**Result type**  comp

**Remarks**  If $(a, m) > 1$ then the value 0 is returned. If $(a, m) = 1$ but $a^c \not\equiv 1 \pmod{m}$ then an error message is printed and the program halts. Since $c$ is factored by trial division, performance will be slow if $c$ has a very large prime factor. In such a case, execution may be interrupted by typing any key.

**Restrictions**  $|a| \leq 10^{18}$, $1 \leq m \leq 10^{18}$, $1 \leq c \leq 10^{18}$

**See also**  PrimRoot

---

**Phi function**  NoThy

**Function**  Computes the Euler phi function $\phi(n)$.

**Declaration**  phi(n: comp)

**Result type**  comp

**Remarks**  Since $n$ is factored by trial division, performance will be slow if $n$ has a very large prime factor. In such a case, execution may be interrupted by typing any key.
### Power function

**Function**
Given $a$, $k$, and $m$, returns $c$ such that $c \equiv a^k \pmod{m}$ and $0 \leq c < m$.

**Declaration**
```
power(a, k, m: comp)
```

**Result type**
`comp`

**Restrictions**
$|a| \leq 10^{18}$, $0 \leq k \leq 10^{18}$, $1 \leq m \leq 10^{18}$

### PrimRoot function

**Function**
Given an integer $a$ and a prime number $p$, returns the least primitive root $g$ of $p$ such that $g > a$.

**Declaration**
```
primroot(p, a: comp)
```

**Result type**
`comp`

**Remarks**
Since $p - 1$ is factored by trial division, performance will be slow if $p - 1$ has a very large prime factor. In such a case, execution may be interrupted by typing any key.

**Restrictions**
$|a| \leq 10^{18}$, $2 \leq p < 10^{18}$

### ReadTimer procedure

**Function**
Give the elapsed time since the timer was set.

**Declaration**
```
readtimer
```

**Remarks**
The elapsed time is stored in the variable TimerString, which is defined to be of type string[35]. The timer must be set before it can be read, by using the procedure SetTimer. Any program employing the timer must declare the unit DOS in the uses statement.

**Restrictions**
The TimerString records only hours, minutes and seconds. If a program runs for more than 24 hours, the number of days must be added to the stated time.

**See also**
SetTimer

**Examples**
See the files slowgcd.pas, factor.pas.
<table>
<thead>
<tr>
<th>Function</th>
<th>Sets the timer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>settimer</td>
</tr>
<tr>
<td>See also</td>
<td>ReadTimer</td>
</tr>
</tbody>
</table>

### SPsP function

<table>
<thead>
<tr>
<th>Function</th>
<th>Applies the strong pseudoprime test base $a$ to $m$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>spsp($a$, $m$: comp)</td>
</tr>
<tr>
<td>Result type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Remarks</td>
<td>If $m$ is proved to be composite then the value False is returned; otherwise</td>
</tr>
<tr>
<td></td>
<td>the calculation is consistent with the hypothesis that $m$ is prime, and</td>
</tr>
<tr>
<td></td>
<td>the value True is returned.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>$</td>
</tr>
</tbody>
</table>

### SqrtModP function

<table>
<thead>
<tr>
<th>Function</th>
<th>Given an integer $a$ and a prime number $p$, returns the number $x$ such</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>that $x^2 \equiv a \pmod{p}$, $0 \leq x \leq p/2$.</td>
</tr>
<tr>
<td>Declaration</td>
<td>sqrtmodp($a$, $p$: comp)</td>
</tr>
<tr>
<td>Result type</td>
<td>comp</td>
</tr>
<tr>
<td>Remarks</td>
<td>If $p$ is found to be composite, or if $a$ is a quadratic nonresidue of $p$,</td>
</tr>
<tr>
<td></td>
<td>then an error message is printed and the program halts.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>$</td>
</tr>
</tbody>
</table>