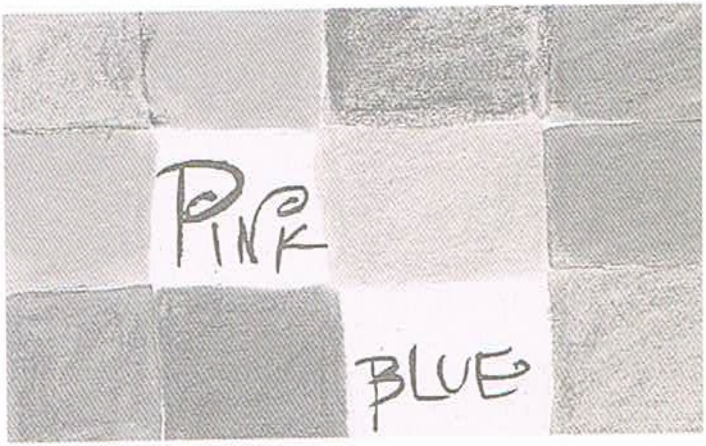
Programming Journal
1999 Yellow Patchwork

By Charlote Greenwood

Date written from 1999 to 2000

My Journal



Property of:

Axel

@ - location URL for item
∴ appended after the Axel item.

_ - Varible
∴ example of varables.

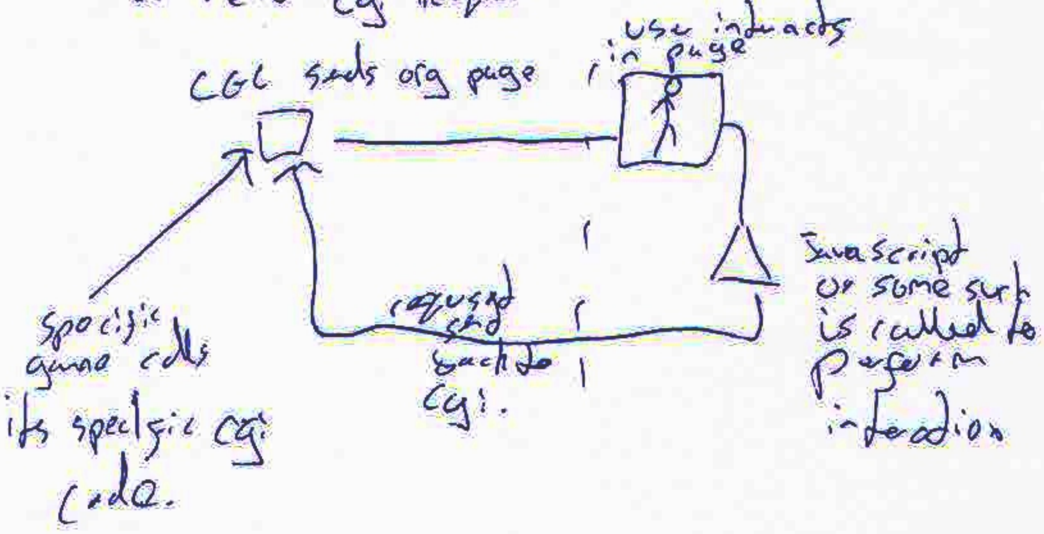
_text (some varable)
_text@localhost
_text@

: axelprog1

remote servers use http to transfer the content of the varable from th base common path /axelp therefore te path /axel-p must be avaible for http on te web domain. The colon cammand adds anothe path to thies root. So :axelprog1 would be resoveld to http:
axel-p/axelprog1/ and then th file 'text' is brought back.

Note that web based varibles are read only e word matches to produce a phrase.

⑧ for all games a CGI script will send the page to the user, where interaction will take place and then this will call up a new cgi request.


CGI sends org page
user interacts in page
Javascript or some such is called to perform interaction
request sent back to cgi.
specific game calls its specific cgi code.

treenix datatypes

byte - 1 Byte
short - 2 Bytes
int - 4 Bytes
dint - 8 Bytes
gint - 16 Bytes (?)

- char
- String

{ defined by location of user }

- single char is either 1 Byte or 2 Bytes.
- Character is from ASCII IBM(a) extended table, or UNICODE(a).

the above are all integers, floating points ~~are~~ use ~~the~~ a stadard integer head with the precision added afterward.

Floating point

m32int
m64int
m128int
m256int

float8
float16
float32

{ 2 Byte 1Byte } 0.0 → 65535.255

This would resolve to:

0.0 → 65535.99

- as math would not be useable with 255 turning around.

∴ Precision

1 Byte = 2 decimal places
2 Byte = 4 decimal places
4 Byte = 9 decimal places
8 Byte = 19 decimal places
& 16 Byte = 38 decimal places
& 32 Byte = 76
& 64 Byte = 152
& 128 Byte = 304
& 256 Byte = 608 decimal places

variable _b

_a != _b

'NOT' condition operator

variable _a

& the processing of the above statement calls the 'NOT' sub with the values of '_a' and '_b'. The sub compares these and the TRUE/FALSE value is placed in the current _condition system variable.

( ) brackets within a condition seperate it's test operators.

© a test sub is like any other runstat sub. It processes it's given information and then returns it's derived conditional true/false in the cwout structure.

Callers such as ~~Wend~~ While / If make use of the raw conditional data.

A single conditional sub processes any conditional argument, both simple and compound for and statement. The statement calls this major sub and then just checks the result.

10 / 100 MPS PCMCIA CARD

00 E0 98 71 49 27

I/O 300 H .

INT 10 H

Treenie

'{' and '}' group statements that follow.

- a special sub is used to search for the closing '}' bracket.

'(' group computationals.

'(' marks the start of a conditional if/while/etc... look for the '(' after them(?).

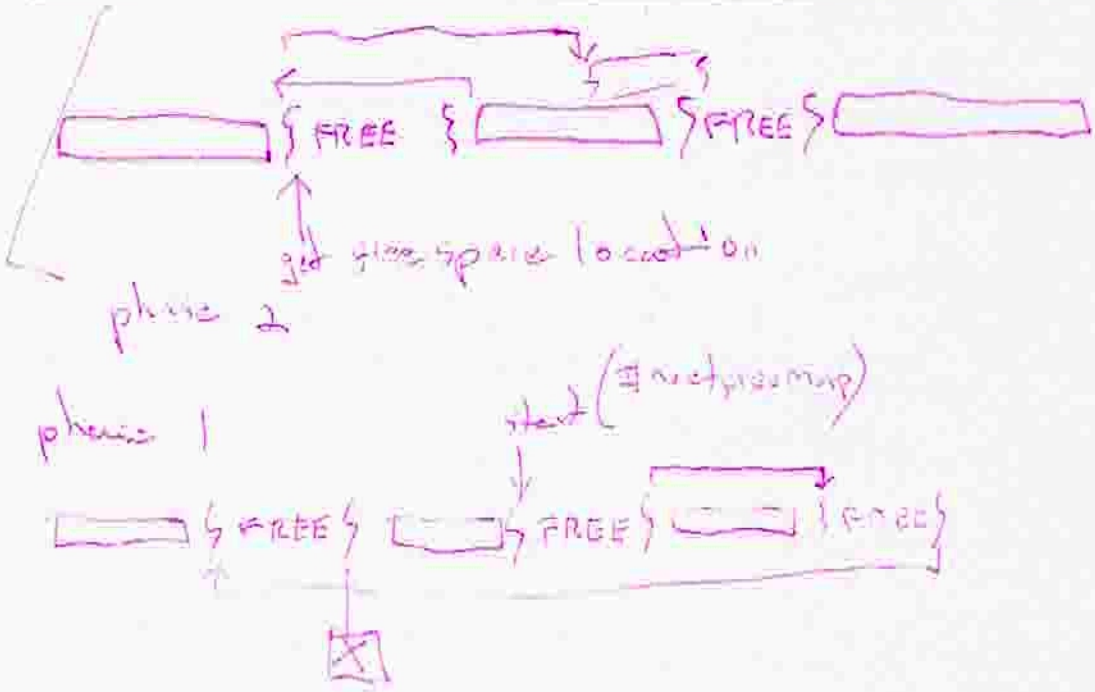
||/&& use the brackets to skip/return from a section.

(~ ~ some statement condition) = returns true or false to the previous, so a group of conditions can be tested and resolved to one vale.

⊘ Variables

all variables **must** have an allocation within the [UV] name store.

allocation 2 - can not be stored.
allocation 3. system argument / temporary store that is erased/freed at the end of processing of the line.



Teeenie Beyond! 32 Bid Numerics

Those are integers that can be signed ~~or~~ or unsigned.

64	Bid	-	(long) dint
128	Bid	-	mega
256	Bid	-	~~kera~~ giga
512	Bid	-	~~[illegible]~~ tera
1024	Bid	-	uldra

None of these are handled by the system's buidt in math functions. They do assume a 32 bid integer math processing capalilidy is present however.

These large values must have the following functions:

- additions
- subtractions
- multiply
- division
- print these values to a string

these are then called by higher functions.

[16 Bid] [16 Bid]
32 Bid

Multiplication

10AB
~~00~~ 2~~0~~32 FF x

10 AB
0020 x
21560

2
32FF x
~~6F~~ 65FE

10 AB
32FF
~~32500~~
3520055

65FE
~~32~~
352 x
~~6950~~

:: => ~~6950~~ 0055 ?

& ANS = 218B2 0055

65FE0
352

MULTIPLICATION

10AB
2032FF x

10AB
20 x
21560
HB

10 AB
32FF x
~~352~~
352 0055
LB

∴ 21560
352 +
218B2

=> 10AB
2032FF x
218B2 0055

30321
A0000 x

197409
655360 x

1293739622..

FFFF again.

30000
A0000 FFFF

3
A
1E
(30)

321
A
1F4A
(8010)

3AAAC (240300)

196608
655360

1288490189

30000
A00000 = (3 x A) x FFFF x FFFF =

10230321
A01092CD x

A010
1023 *

A16E230
+ [16 BIT SHIFT]
A16E230
→0000 → 0000

∴

~~(3 x 10) x 65535 x 65535 =~~
~~128845086750~~

3932130

check

~~194~~ 196608
655360 x

128849018880

(~~3 x 10~~) x 6

(3 x 10) x 65536 x 65536 = 128849018880

[A]0000 [3]0000

∴ (3 x 65536) x (10 x 65536)

1E00000000
lower upper

9 8 7 6 5 4 3 $^{10^2}_{10}$ 1

7017263,95, = ~~1024120093~~ 29D37EBB

get length of string numeric (after parsing)

get first digit $^{10 \times Pos)}$

7000,000,00 = [29B9][B700]

65535.

1

70172.

6395

701726395 =

7^{10^8}

0^{10^7}

1^{10^6}

7^{10^5}

2^{10^4}

6^{10^3}

3^{10^2}

9^{10^1}

5^{10^0}

(D)

1000 = 3E8

10000 = 2710

100000 = 186A0

1000000 = F4240

Display of numbers.

```
  256
  256 ×
 ------
     0
    0
 16 36
  4 3
```

```
  256
  256 ×
 ------
 12
 10
  4
 ---
 26
  30
  25
  10
 ----
  3  6
  3  0
  1  2
```

65536

```
  256
  256
 ------
 40000
  1000
  1200
 10000
  2500
   300
  1200
   300
```

40

5 5536

1

Divisio.

$$\frac{\begin{array}{r}\text{FEAB10} \\ \text{30AC} \div\end{array}}{} = \text{53B}$$

30AC ⟌FEAB10   31

1D4

30 ⟌FE   AC ⟌EAB10

50

E

E

FE / 30 = 5

B

AB10 / 30 =

~~30AC ⟌FE93A4 = 53B~~

~~-5~~

5

30AC ⟌FE93A4

F0

E93

num * 256 (+) → 2odad

3850
50
77
= F ⊘ A
32
= 15 10
50

15 10
50
= 5

~~5. 12~~

5 12
10
5 22

15
5
3

15 × ~~5~~ = 75
- 77
2. => 10
12
22

A^2 A^1
B^1
=

145 & 1 & (38)(4) (B^1 / 256)
= 145 & 1 & 4
1D4C = 150 ✓

$(256 / B^1) = R^1$

$R^1 * A^2 = C^1$

7500
50
= 150

1D 4C
29 76

76/50
= 1 & 12
r 26 (3) 38

5 × 29

26
12 &

$C^1 = C^1 + (A^1 / B^1)$
& (Remainder of A^1/B^1)
& Remainder of
$(256 / B^1)$.

81050
50
1621

= 1 3C 9A
32

A^3 A^2 A^1
= 1 60 154

50 B^1

3D

= 256/50 = 5.12

$A^3 * 5 = 5^{R12}$ = 6

56

$(A^2$ ~~+12~~$) * 5 =$ ~~360~~ $+12 = 312R^{12}$ / 256 = 1 => ~~104~~

~~154 (A^1 + 12) / 50 =~~

$(A^1 / B^1) = 154 / 50$ 3^{R8} + R+8 (20) = 5

6 104+5

6 109 = 66D ? (+24?

1645. ?

(-41).

$7500 \overline{)}\ 50$ Decimal

Division 2

converd do n^{256}

$A^2 \quad A^1$

$\underline{\quad B^1}$

$\left(A^1 / B^1\right)$ ⇒

L_{rm^1}

$$\left[\left(256 / B^1\right) * A^2 + \left(A^1 / B^1\right) + rm^1 + r \right.$$

$L_{cm^2} \qquad L_{rm^2}$

$256 / \overset{B^1}{50} = 5^{R12}$

~~60*~~

$A^3 * 5 = 5^{R12} \ (+1) = 6$

$A^2 * 5 = 300 + 12 = 312^{R12} = 1,56$

$A^1 / B^1 = 154 / 50 = 3.^{R4}$

$56 + 3 + 12$

$56 + 3 + (?). = 1595 \ (-1621) = 26.$

Division - 1

```
F   A              3850
16  10              513
2   1             -------
--------          7.50...
8   10

16  5    5         150      1500
                    10        10
                  -----     ------
                    15       150

5DC      =>   5   220        => 150
 .A                10
-----         -----------
              ~~2~~   22

Ex                 128+22:150.

5    220
↓     10
[5x256] → ~~added to next col~~
↓
```

$12460 \overline{)16683940}$

65536

```
  256 (A)
  256 (B)
 ------
 40000
 10000
  1200
 ------
 10000
  2500
   300
 ------
  1200
   300
    36
 ------
 65536
 1
```

count (B) 2 spaces from right
+ count (A) 2 spaces from right
= 4 padded zeros.

[2×2] = 4

4 padding

∴ 40000

Each column is its own u ~~short~~ char store.

∴ 4 would be stored in u~~short~~ char [4] (~~[illegible]~~ padding - 1)

at the end of the multiply run. each ushort is run through ~~left~~ right to left. The <10 digit is taken and the MSD is passed (added to) the next ushort. What's left at the end is put into the string.

MULTIPLICATION

A10C D301
19B89CD2 ×

2701972225
431529170

701726395

~~19B89~~ ×

19B8
A10C ×

SHIFT LEFT 32 ~~16~~ BITS &
(65536 × 65536)

19B8
D301 ×

SHIFT LEFT 16 BITS & ..
and add coloms (65536)

9CD2 ×
A10C

SHIFT LEFT 16 BITS
and add colons

9CD2
D301 ×

NO SHIF
-add coloms

↓
ANS

MULTIPLICATION (2)

10230321
A01042CD x

= 1023
A010 x

A16 E230

0321
A010 x

1FAD 210

1023
42CD

940E207

0321
92CD

[1CB]536D

A16E230
940E207 x

210730017
268 5440717 x

7270294411^{1017}

A17017D

30321
A92CD x

3
A

1E

32
A

1F4A

3
92CD

1B867

321
92CD

[1CB]536D

197409
692941 x

13679278 99...

Tests Addition

FF[EA 00][10 AB] =
+ [20[32 FF]

10AB
20 32 FF +
2043AA. ?

~~4269~~

4267
13055
2114474

4267
2114474

FFEA

4267
13055
17322

4267
2110207 .
2114474

16445439
1092592 +
17538031

FA EFFF
10ABF0 +

= 10 B9BEF
(17538031)

EFFF
ABF0
[1] 9BEF

→ 00FA+1 ⟵ 00FB

∴ FB
10
10B ⟹ 10B9BEF

☒ variable conversion.. (ints)

signed / unsigned

2nd ~~1st~~ phase

converts type to ulong - by converting with the sign

1st ~~2nd~~ phase

determines the signed status from (prepop [0] & 1)

3rd phase

ulong is converted to the dest type

- a user object can be created.

- it may use 'system nodes', values assigned from system resources to the user component.

- where possible, these may be set within an user object using the 'system node methods'.

- new object [illegible] $name~~(format(.$name...~~,

primary

provides a framework to hang the user's object to.

new object $name

extend $name

$name.free()

new [public] [[fixed]]
[group]

~~extend~~ [illegible] [illegible]

primary

extend $name

primary

string8 - Romanized text

string16 - Unicode

condition testing is a built in process that return a true/false value when using the test operators.

condition

for

while ( ) {

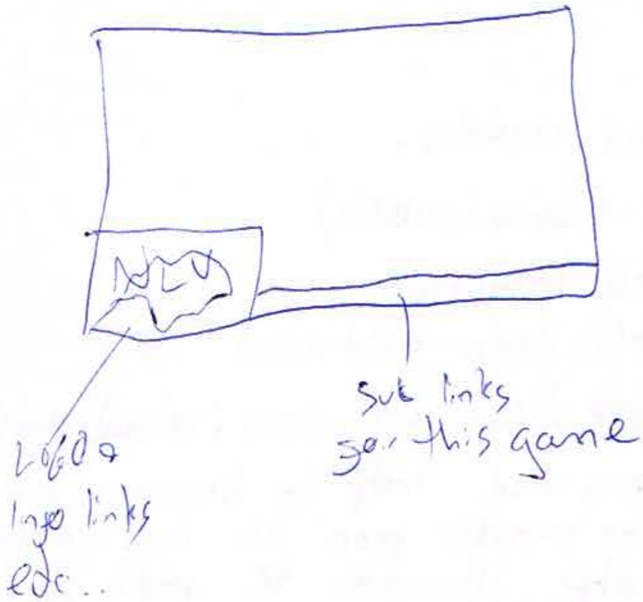
execution is branched here

} — Do statements in brackets while condition is true

③ - this final bracket should be indexed at start to save un needed processing.

{ Bracket is linked to the connected statement

the presence of a bracket '{' causes the appropiate token to be stored and the runstat will branch after testing the condition is true and execute the statements until another '}' is come accross.


Sub links
for this game
LOGO &
logo links
etc...

appended pages

```
//
// long/mega/giga/tera/ulta number display
// written by Charlote Greenwood
//
// updated : june 28 2001
//
// +++++++++++++++++++++++++++++++++++++++++++
//                         written on bsd unix
// -------------------------------------------
// Copyright 2001 Charlote Elizabeth Greenwood
//
// This software is provided as-is without warranty
// or claim for having suitability for any purpose
// what-so-ever and is provided for informational
// purposes only. If you use this software you are
// agreeing you do so at you're own risk.
//
// You may copy and redistribute the material in any medium or
// format. Under the following terms:
// Attribution - You must give appropriate credit, and indicate
// if changes were made. You may do so in any reasonable manner,
// but not in any way that suggests endorsement of you or your
// use.
// NonCommercial - You may not use the material for commercial
// purposes. NoDerivatives - If you remix, transform, or build
// upon the material, you may not distribute the modified
// material.
// No additional restrictions - You may not apply legal terms or
// technological measures that legally restrict others from
// doing anything this license permits.
// -------------------------------------------

#include<stdio.h>

unsigned short maxtnxnumbits=1024;
typedef struct {
unsigned short length;
unsigned char flags;
unsigned char data[1024];
}treebase;

int tnxnummake(treebase *_tnxnum, unsigned short _bits)
{
if (_bits>maxtnxnumbits) return(-1);
unsigned short _ushort;
_tnxnum->length=_bits;
for (_ushort=0; _ushort<_bits; _ushort++) {
_tnxnum->data[_ushort]=0;
}
_tnxnum->flags=0;
return(0);
}
```

```
int tnxnummake64(treebase *_tnxnum)
{
return(tnxnummake(_tnxnum, 64));
}

int tnxnummake128(treebase *_tnxnum)
{
return(tnxnummake(_tnxnum, 128));
}

int tnxnummake256(treebase *_tnxnum)
{
return(tnxnummake(_tnxnum, 256));
}

int tnxnummake512(treebase *_tnxnum)
{
return(tnxnummake(_tnxnum, 512));
}

int tnxnummake1024(treebase *_tnxnum)
{
return(tnxnummake(_tnxnum, 1024));
}

void tnxnumzero(treebase *_dintout)
{
unsigned short _ushort;
for (_ushort=0; _ushort<_dintout->length; _ushort++) {
_dintout->data[_ushort]=0;
}
}

int tnxnumsetfromrawuint(treebase *_dint, unsigned int _uint)
{
char _a;
int _dpos=_dint->length-1;
unsigned int _b=255, _c=1;
if (_dint->length<4) return(-1);
for (_a=0; _a<4; _a++, _dpos--) {
_dint->data[_dpos]=(_uint&_b)/_c;
_b*=256;
_c*=256;
}
while (_dpos!=-1) {
_dint->data[_dpos]=0;
_dpos--;
}
return(0);
}
```

```
int tnxnumsetfromuint(treebase *_dint, unsigned int _uint)
{
_dint->flags&=254;
return(tnxnumsetfromrawuint(_dint, _uint));
}

int tnxnumsetfromint(treebase *_dint, int _int)
{
unsigned int _uint;
if (_uint<0) {
_dint->flags|=1;
_int=-_int;
}else {
_dint->flags&=254;
}
_uint=(unsigned int)_int;
return(tnxnumsetfromrawuint(_dint, _uint));
}

int tnxnummul(treebase *_dinta, treebase *_dintb, treebase *_dintout)
{
int _a, _b, _c, _e=_dintout->length;
unsigned short _cols[_dintout->length], _mresult, _carry;
for (_carry=0; _carry<_dintout->length; _carry++) {
_cols[_carry]=0;
}
if ((_dinta->flags&1)==1) {
if ((_dintb->flags&1)==1) {
_dintout->flags&=254;
}else {
_dintout->flags|=1;
}
}else {
if ((_dintb->flags&1)==1) {
_dintout->flags|=1;
}else {
_dintout->flags&=254;
}
}
for (_b=_dintb->length; _b>0; _b--) {
_c=_dintout->length-(_b+_dinta->length);
for (_a=_dinta->length; _a>0; _a--, _c--) {
_mresult=_dinta->data[_a]*_dintb->data[_b];
if (_mresult>0) {
if (_c<1) {
for (_c=0; _c<_dintout->length; _c++) {
_dintout->data[_c]=255;
}
return(0);
}
_e=_c-1;
_cols[_e]+=_dinta->data[_a]*_dintb->data[_b];
}
}
}
```

```
_carry=0;
for (_c=_dintout->length; _c>_e; _c--) {
if (_cols[_c]>255) {
if (_c==0) {
for (_a=0; _a<_dintout->length; _a++) {
_dintout->data[_a]=255;
}
return(0);
}
_carry=_cols[_c]/256;
_cols[_c-1]+=_carry;
_dintout->data[_c]=(unsigned char)(_cols[_c]-(_carry*256));
}else {
_carry=0;
_dintout->data[_c]=(unsigned char)_cols[_c];
}
_cols[_b]=0;
}
return(0);
}

int tnxnumtotext(treebase *_dintin, char *_textout, unsigned int *_textlength)
{
int _a, _b, _c, _d, _e, _val1, _carry;
char _textnum[2468], _thisnum[2468];
unsigned char _cols[2468];
for (_a=0; _a<2467; _a++) {
_textnum[_a]=48;
_cols[_a]=0;
}
_textnum[2467]=0, _cols[2467]=0;
for (_a=0, _e=2467;_a<_dintin->length;_a++) {
if (_a!=0) {
for (_d=_e; _d<2467; _d++) {
_cols[_d-2]+=(((int)_textnum[_d])-48)*2;
}
for (_d=_e; _d<2467; _d++) {
_cols[_d-1]+=(((int)_textnum[_d])-48)*5;
}
for (_d=_e; _d<2467; _d++) {
_cols[_d-0]+=(((int)_textnum[_d])-48)*6;
}
_carry=0;
for (_b=2466;(_b>_e-3 || _cols[_b]!=0) && _b>-1; _b--) {
if (_cols[_b]>9) {
if (_b==0) {
return(-1);
}
_carry=_cols[_b]/10;
_cols[_b-1]+=_carry;
_textnum[_b]=(char)(48+(_cols[_b]-(_carry*10)));
}else {
_carry=0;
_textnum[_b]=(char)(48+_cols[_b]);
}
_cols[_b]=0;
}
if (_b<_e) _e=_b;
}
_carry=0;
sprintf(_thisnum, "%u", _dintin->data[_a]);
for (_c=strlen(_thisnum)-1, _d=2466; _c>-1; _c--, _d--) {
_val1=((int)_textnum[_d])+((int)_thisnum[_c])+_carry-96;
if (_val1>9) {
_carry=_val1/10;
_textnum[_d]=(char)(48+(_val1-(_carry*10)));
}else {
_carry=0;
_textnum[_d]=(char)(48+_val1);
}
}
addcarry:;
if (_carry>0) {
_val1=((int)_textnum[_d])-48+_carry;
if (_val1>9) {
_carry=_val1/10;
_textnum[_d]=(char)(48+(_val1-(_carry*10)));
}else {
_carry=0;
_textnum[_d]=(char)(48+(_val1));
}
if (_carry>0) {
_d--;
if (_d<0) {
return(-1);
}
goto addcarry;
}
}
if (_d<_e) _e=_d;
}
for (_a=0; _a<2468 && _textnum[_a]=='0'; _a++);
if (*_textlength<(2067-_a)) return(-1);
*_textlength=2067-_a;
strcpy(_textout, &_textnum[_a]);
return(0);
}
```

```
void main(void)
{
int _a;
char _text[2068];
unsigned int _textlength=2068;
treebase _dinta, _dintb, _dintresult;

tnxnummake64(&_dinta);
tnxnummake64(&_dintb);
tnxnummake64(&_dintresult);

tnxnumzero(&_dinta);

tnxnumsetfromuint(&_dinta, 2);
tnxnumsetfromint(&_dinta, -2);
printf("mulitplying..\n");
if (tnxnummul(&_dinta, &_dintb, &_dintresult)==-1) exit(-1);
printf("numtotext..\n");
if (tnxnumtotext(&_dintresult, &_text[0], &_textlength)==-1) exit(-1);


printf("\nnumber is [%s]\n\n", _text);
exit(0);
}
```