(/wiki/Rosetta_Code)

Create account (/mw/index.php?title=Special:UserLogin&returnto=Loops%2FWhile&type=signup)
Log in (/mw/index.php?title=Special:UserLogin&returnto=Loops%2FWhile)

Search

Page (/wiki/Loops/While) Discussion (/wiki/Talk:Loops/While) Edit (/mw/index.php?title=Loops/While&action=edit) History (/mw/index.php?title=Loops/While&action=histo

I'm working on modernizing Rosetta Code's infrastructure. Starting with communications. Please accept this time-limited open invite to RC's Slack. (https://join.slack.com/t/rosettacode/shared_invite/zt-glwmugtu-xpMPcqHs0u6MsK5zCmJF~Q). --Michael Mol (/wiki/User:Short_Circuit) (talk (/wiki/User_talk:Short_Circuit)) 20:59, 30 May 2020 (UTC)

Loops/While

< Loops (/wiki/Loops)

Task

Start an integer value at **1024**.

Loop while it is greater than zero.

Print the value (with a newline) and divide it by two each time through the loop.

(/wiki/Category:Solutions_by

Loops/While
You are encouraged to solve this task (/wiki/Rosetta_Code:Solve_a according to the task description, using any language you may know.

Related tasks

- Loop over multiple arrays simultaneously (/wiki/Loop_over_multiple_arrays_simultaneously)
- Loops/Break (/wiki/Loops/Break)
- Loops/Continue (/wiki/Loops/Continue)
- Loops/Do-while (/wiki/Loops/Do-while)
- Loops/Downward for (/wiki/Loops/Downward_for)
- Loops/For (/wiki/Loops/For)
- Loops/For with a specified step (/wiki/Loops/For_with_a_specified_step)
- Loops/Foreach (/wiki/Loops/Foreach)
- Loops/Increment loop index within loop body (/wiki/Loops/Increment_loop_index_within_loop_body)
- Loops/Infinite (/wiki/Loops/Infinite)
- Loops/N plus one half (/wiki/Loops/N_plus_one_half)
- Loops/Nested (/wiki/Loops/Nested)
- **Loops/While**
- Loops/with multiple ranges (/wiki/Loops/with_multiple_ranges)
- Loops/Wrong ranges (/wiki/Loops/Wrong_ranges)

Contents

0815 (/wiki/Category:0815)

```
<:400:~}:_:%<:a:~$=<:2:=/^:_:
```

11l (/wiki/Category:11l)

Translation of: Python

```
V n = 1024
L n > 0
   print(n)
   n I/= 2
```

360 Assembly (/wiki/Category:360_Assembly)

Basic

Using binary arithmetic. Convert results to EBCDIC printable output.

```
*        While                     27/06/2016
WHILELOO CSECT                     program's control section
         USING WHILELOO,12         set base register
         LR    12,15               load base register
         LA    6,1024              v=1024
LOOP     LTR   6,6                 while v>0
         BNP   ENDLOOP             .
         CVD   6,PACKED              convert v to packed decimal
         OI    PACKED+7,X'0F'        prepare unpack
         UNPK  WTOTXT,PACKED         packed decimal to zoned printable
         WTO   MF=(E,WTOMSG)         display v
         SRA   6,1                   v=v/2   by right shift
         B     LOOP                end while
ENDLOOP  BR    14                  return to caller
PACKED   DS    PL8                 packed decimal
WTOMSG   DS    0F                  full word alignment for wto
WTOLEN   DC    AL2(8),H'0'         length of wto buffer (4+1)
WTOTXT   DC    CL4' '              wto text
         END   WHILELOO
```

Output:

(+ sign indicates "problem state" (non system key) issued WTO's

```
+1024
+0512
+0256
+0128
+0064
+0032
```

Structured Macros

```
*        While                     27/06/2016
WHILELOO CSECT
         USING WHILELOO,12         set base register
         LR    12,15               load base register
         LA    6,1024              v=1024
         DO WHILE=(LTR,6,P,6)      do while v>0
         CVD   6,PACKED              convert v to packed decimal
         OI    PACKED+7,X'0F'        prepare unpack
         UNPK  WTOTXT,PACKED         packed decimal to zoned printable
         WTO   MF=(E,WTOMSG)         display
         SRA   6,1                   v=v/2   by right shift
         ENDDO ,                   end while
         BR    14                  return to caller
PACKED   DS    PL8                 packed decimal
WTOMSG   DS    0F                  full word alignment for wto
WTOLEN   DC    AL2(8),H'0'         length of wto buffer (4+1)
WTOTXT   DC    CL4' '              wto text
         END   WHILELOO
```

Output:

Same as above

6502 Assembly (/wiki/Category:6502_Assembly)

Code is called as a subroutine (i.e. JSR LoopsWhile). Specific OS/hardware routines for printing are left unimplemented.

```
LoopsWhile:     PHA                     ;push accumulator onto stack

                LDA #$00                ;the 6502 is an 8-bit processor
                STA Ilow                ;and so 1024 ($0400) must be stored in two memory locations
                LDA #$04
                STA Ihigh
WhileLoop:      LDA Ilow
                BNE NotZero
                LDA Ihigh
                BEQ EndLoop
NotZero:        JSR PrintI              ;routine not implemented
                LSR Ihigh               ;shift right
                ROR Ilow                ;rotate right
                JMP WhileLoop

EndLoop:        PLA                     ;restore accumulator from stack
                RTS                     ;return from subroutine
```

AArch64 Assembly (/wiki/Category:AArch64_Assembly)

Works with: as (/mw/index.php?title=As&action=edit&redlink=1) version Raspberry Pi 3B version Buster 64 bits

```
/* ARM assembly AARCH64 Raspberry PI 3B */
/*  program loopwhile64.s   */

/*******************************************/
/* Constantes file                         */
/*******************************************/
/* for this file see task include a file in language AArch64 assembly*/
.include "../includeConstantesARM64.inc"
/*********************************/
/* Initialized data              */
/*********************************/
.data
szMessResult:      .asciz "@"            // message result
szCarriageReturn:  .asciz "\n"
/*********************************/
/* UnInitialized data            */
/*********************************/
.bss
sZoneConv:           .skip 24
/*********************************/
/*  code section                 */
/*********************************/
.text
.global main
main:                                    // entry of program
    mov x20,#1024                        // loop counter
1:                                       // begin loop
    mov x0,x20
    ldr x1,qAdrsZoneConv                 // display value
    bl conversion10                      // decimal conversion
    ldr x0,qAdrszMessResult
    ldr x1,qAdrsZoneConv
    bl strInsertAtCharInc                // insert result at @ character
    bl affichageMess                     // display message
    ldr x0,qAdrszCarriageReturn
    bl affichageMess                     // display return line
    lsr x20,x20,1                        // division by 2
    cmp x20,0                            // end ?
    bgt 1b                               // no ->begin loop one


100:                                     // standard end of the program
    mov x0,0                             // return code
    mov x8,EXIT                          // request to exit program
    svc 0                                // perform the system call

qAdrsZoneConv:            .quad sZoneConv
qAdrszMessResult:         .quad szMessResult
qAdrszCarriageReturn:     .quad szCarriageReturn

/********************************************************/
/*        File Include fonctions                        */
/********************************************************/
/* for this file see task include a file in language AArch64 assembly */
.include "../includeARM64.inc"
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

ActionScript (/wiki/Category:ActionScript)

```
var i:int = 1024;
while (i > 0) {
    trace(i);
    i /= 2;
}
```

Ada (/wiki/Category:Ada)

```
declare
   I : Integer := 1024;
begin
   while I > 0 loop
      Put_Line(Integer'Image(I));
      I := I / 2;
   end loop;
end;
```

Agena (/wiki/Category:Agena)

Tested with Agena 2.9.5 Win32

```
scope
    local i := 1024;
    while i > 0 do
        print( i );
        i := i \ 2
    od
epocs
```

Aime (/wiki/Category:Aime)

```
integer i;

i = 1024;
while (i) {
    o_plan(i, "\n");
    i /= 2;
}
```

ALGOL 60 (/wiki/Category:ALGOL_60)

The Loops/While structure was in the Algol 60 report of January 1963.

```
begin
        comment Loops/While - algol60 - 21/10/2014;
        integer i;
        for i:=1024,i div 2 while i>0 do outinteger(1,i)
end
```

Output:

```
1024  512  256  128  64  32  16  8  4  2  1
```

ALGOL 68 (/wiki/Category:ALGOL_68)

Works with: ALGOL 68 (/wiki/ALGOL_68) version Revision 1 - no extensions to language used
Works with: ALGOL 68G (/wiki/ALGOL_68G) version Any - tested with release 1.18.0-9h.tiny (https://sourceforge.net/projects/algol68/files/algol68g/algol68g-1.18.0/algol68g-1.18.0-9h.tiny.el5.centos.fc11.i386.rpm/download)
Works with: ELLA ALGOL 68 (/wiki/ELLA_ALGOL_68) version Any (with appropriate job cards) - tested with release 1.8-8d (https://sourceforge.net/projects/algol68/files/algol68toc/algol68toc-1.8.8d/algol68toc-1.8-8d.fc9.i386.rpm/download)

```
INT i := 1024;
WHILE i > 0 DO
   print(i);
   i := i OVER 2
OD
```

Output:

```
      +1024       +512       +256       +128        +64        +32        +16         +8         +4         +2         +1
```

ALGOL W (/wiki/Category:ALGOL_W)

```
begin
    integer i;
    i := 1024;
    while i > 0 do
    begin
        write( i );
        i := i div 2
    end
end.
```

ALGOL-M (/wiki/Category:ALGOL-M)

```
begin
    integer i;
    i := 1024;
    while i > 0 do begin
        write( i );
        i := i / 2;
    end;
end
```

AmbientTalk (/wiki/Category:AmbientTalk)

Note: in AmbientTalk, while:do: is a keyworded message (as in Smalltalk). Both arguments to this message must be blocks (aka anonymous functions or thunks).

```
// print 1024 512 etc
def i := 1024;
while: { i > 0 } do: {
  system.print(" "+i);
  i := i/2;
}
```

AmigaE (/wiki/Category:AmigaE)

```
PROC main()
  DEF i = 1024
  WHILE i > 0
    WriteF('\d\n', i)
    i := i / 2
  ENDWHILE
ENDPROC
```

AppleScript (/wiki/Category:AppleScript)

AppleScript does not natively support a standard out. Use the Script Editor's Event Log as the output.

```
set i to 1024
repeat while i > 0
        log i
        set i to i / 2
end repeat
```

Applesoft BASIC (/wiki/Category:Applesoft_BASIC)

```
 10 I% = 1024
 20  IF I% > 0 THEN  PRINT I%:I% = I% / 2: GOTO 20
```

ARM Assembly (/wiki/Category:ARM_Assembly)

Works with: as (/mw/index.php?title=As&action=edit&redlink=1) version Raspberry Pi

```

/* ARM assembly Raspberry PI  */
/*  program loopwhile.s   */

/* Constantes    */
.equ STDOUT, 1     @ Linux output console
.equ EXIT,   1     @ Linux syscall
.equ WRITE,  4     @ Linux syscall

/*********************************/
/* Initialized data              */
/*********************************/
.data
szMessResult:      .ascii ""                    @ message result
sMessValeur:       .fill 11, 1, ' '
szCarriageReturn:  .asciz "\n"
/*********************************/
/* UnInitialized data            */
/*********************************/
.bss
/*********************************/
/*  code section                 */
/*********************************/
.text
.global main
main:                                       @ entry of program
    mov r4,#1024                            @ loop counter
1:                                          @ begin loop
    mov r0,r4
    ldr r1,iAdrsMessValeur                  @ display value
    bl conversion10                         @ decimal conversion
    ldr r0,iAdrszMessResult
    bl affichageMess                        @ display message
    ldr r0,iAdrszCarriageReturn
    bl affichageMess                        @ display return line
    lsr r4,#1                               @ division by 2
    cmp r4,#0                               @ end ?
    bgt 1b                                  @ no ->begin loop one


100:                                        @ standard end of the program
    mov r0, #0                              @ return code
    mov r7, #EXIT                           @ request to exit program
    svc #0                                  @ perform the system call

iAdrsMessValeur:          .int sMessValeur
iAdrszMessResult:         .int szMessResult
iAdrszCarriageReturn:     .int szCarriageReturn
/******************************************************************/
/*     display text with size calculation                        */
/******************************************************************/
/* r0 contains the address of the message */
affichageMess:
    push {r0,r1,r2,r7,lr}                   @ save  registres
    mov r2,#0                               @ counter length
1:                                          @ loop length calculation
    ldrb r1,[r0,r2]                         @ read octet start position + index
    cmp r1,#0                               @ if 0 its over
    addne r2,r2,#1                          @ else add 1 in the length
    bne 1b                                  @ and loop
                                            @ so here r2 contains the length of the message
    mov r1,r0                               @ address message in r1
    mov r0,#STDOUT                          @ code to write to the standard output Linux
    mov r7, #WRITE                          @ code call system "write"
    svc #0                                  @ call systeme
    pop {r0,r1,r2,r7,lr}                    @ restaur registers */
    bx lr                                   @ return
/******************************************************************/
/*     Converting a register to a decimal                                 */
/******************************************************************/
/* r0 contains value and r1 address area   */
.equ LGZONECAL,   10
conversion10:
    push {r1-r4,lr}                         @ save registers
    mov r3,r1
    mov r2,#LGZONECAL
1:                                          @ start loop
    bl divisionpar10                        @ r0 <- dividende. quotient ->r0 reste -> r1
    add r1,#48                              @ digit
    strb r1,[r3,r2]                         @ store digit on area
    cmp r0,#0                               @ stop if quotient = 0
    subne r2,#1                               @ previous position
    bne 1b                                  @ else loop
```

```
                                            @ end replaces digit in front of area
    mov r4,#0
2:
    ldrb r1,[r3,r2]
    strb r1,[r3,r4]                         @ store in area begin
    add r4,#1
    add r2,#1                               @ previous position
    cmp r2,#LGZONECAL                       @ end
    ble 2b                                  @ loop
    mov r1,#0                               @ final zero
    strb r1,[r3,r4]
100:
    pop {r1-r4,lr}                          @ restaur registres
    bx lr                                   @return
/***************************************************/
/*   division par 10   signé                       */
/* Thanks to http://thinkingeek.com/arm-assembler-raspberry-pi/*
/* and   http://www.hackersdelight.org/            */
/***************************************************/
/* r0 dividende   */
/* r0 quotient */
/* r1 remainder  */
divisionpar10:
  /* r0 contains the argument to be divided by 10 */
    push {r2-r4}                           @ save registers  */
    mov r4,r0
    mov r3,#0x6667                         @ r3 <- magic_number  lower
    movt r3,#0x6666                        @ r3 <- magic_number  upper
    smull r1, r2, r3, r0                   @ r1 <- Lower32Bits(r1*r0). r2 <- Upper32Bits(r1*r0)
    mov r2, r2, ASR #2                     @ r2 <- r2 >> 2
    mov r1, r0, LSR #31                    @ r1 <- r0 >> 31
    add r0, r2, r1                         @ r0 <- r2 + r1
    add r2,r0,r0, lsl #2                   @ r2 <- r0 * 5
    sub r1,r4,r2, lsl #1                   @ r1 <- r4 - (r2 * 2)  = r4 - (r0 * 10)
    pop {r2-r4}
    bx lr                                  @ return
```

ArnoldC (/wiki/Category:ArnoldC)

```
IT'S SHOWTIME
HEY CHRISTMAS TREE n
YOU SET US UP 1024
STICK AROUND n
TALK TO THE HAND n
GET TO THE CHOPPER n
HERE IS MY INVITATION n
HE HAD TO SPLIT 2
ENOUGH TALK
CHILL
YOU HAVE BEEN TERMINATED
```

Arturo (/wiki/Category:Arturo)

```
i: 1024

while [i>0] [
        print i
        i: i/2
]
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

AutoHotkey (/wiki/Category:AutoHotkey)

```
i = 1024
While (i > 0)
{
  output = %output%`n%i%
  i := Floor (http://www.autohotkey.com/docs/Functions.htm#BuiltIn)(i / 2)
}
MsgBox (http://www.autohotkey.com/docs/commands/MsgBox.htm) % output
```

AWK (/wiki/Category:AWK)

```
BEGIN {
  v = 1024
  while(v > 0) {
    print v
    v = int(v/2)
  }
}
```

Axe (/wiki/Category:Axe)

```
1024→A
While A>0
 Disp A►Dec,i
 A/2→A
End
```

BASIC (/wiki/Category:BASIC)

Works with: QuickBasic (/wiki/QuickBasic) version 4.5

```
i = 1024
WHILE i > 0
   PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) i
   i = i / 2
WEND
```

BaCon (/wiki/Category:BaCon)

```

i = 1024
WHILE i > 0
   PRINT i
   i = i / 2
WEND
```

Commodore BASIC (/wiki/Category:Commodore_BASIC)

There is no WHILE construct in Commodore BASIC. A GOTO construct is used instead. Also, an integer variable name has a % sign as its suffix.

```
10 N% = 1024
20 IF N% = 0 THEN 60
30 PRINT N%
40 N% = N%/2
50 GOTO 20
60 END
```

BBC BASIC (/wiki/Category:BBC_BASIC)

Works with: BBC BASIC for Windows (/wiki/BBC_BASIC_for_Windows)

```
      i% = 1024
      WHILE i%
        PRINT i%
        i% DIV= 2
      ENDWHILE
```

IS-BASIC (/wiki/Category:IS-BASIC)

```
100 LET I=1024
110 DO WHILE I>0
120   PRINT I
130   LET I=IP(I/2)
140 LOOP
```

BASIC256 (/wiki/Category:BASIC256)

```
i = 1024

while i > 0
        print i
        i = i \ 2
end while

end
```

bc (/wiki/Category:Bc)

```
i = 1024
while (i > 0) {
    i
    i /= 2
}
```

Befunge (/wiki/Category:Befunge)

```
84*:*>       :v
     ^/2,*25.:_@
```

blz (/wiki/Category:Blz)

```
num = 1024
while num > 1 # blz will automatically cast num to a fraction when dividing 1/2, so this is necessary to stop an infinite loop
    print(num)
    num = num / 2
end
```

Bracmat (/wiki/Category:Bracmat)

```
1024:?n & whl'(!n:>0 & out$!n & div$(!n.2):?n)
```

Brat (/wiki/Category:Brat)

Converts to integers so output is a little bit shorter and neater.

```
i = 1024
while { i > 0 } {
    p i
    i = (i / 2).to_i
}
```

C (/wiki/Category:C)

```
int i = 1024;
while(i > 0) {
  printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d\n", i);
  i /= 2;
}
```

In for loop fashion:

```
int i;
for(i = 1024;i > 0; i/=2){
   printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d\n", i);
}
```

C# (/wiki/Category:C_sharp)

```
int i = 1024;
while(i > 0){
   System.Console.WriteLine(i);
   i /= 2;
}
```

C++ (/wiki/Category:C%2B%2B)

```
int i = 1024;
while(i > 0){
  std::cout << i << std::endl;
  i /= 2;
}
```

Alternatively, it can be done with `for`:

```
for(int i = 1024; i > 0; i /= 2)
  std::cout << i << std::endl;
```

Instead of `i /= 2` one can also use the bit shift operator `i >>= 1` on integer variables.

Indeed, in C++,

```
for(init; cond; update){
  statement;
}
```

is equivalent to

```
{
  init;
  while(cond){
    statement;
    update;
  }
}
```

Caché ObjectScript (/wiki/Category:Cach%C3%A9_ObjectScript)

```
WHILELOOP
    set x = 1024
    while (x > 0) {
        write x,!
        set x = (x \ 2)    ; using non-integer division will never get to 0
    }

    quit
```

Output:

```
SAMPLES>DO ^WHILELOOP
1024
512
256
128
64
32
16
8
4
2
1
```

Chapel (/wiki/Category:Chapel)

```
var val = 1024;
while val > 0 {
        writeln(val);
        val /= 2;
}
```

ChucK (/wiki/Category:ChucK)

```

1024 => int value;

while(value > 0)
{
    <<<value>>>;
    value / 2 => value;
}
```

Clojure (/wiki/Category:Clojure)

```
(def i (ref 1024))

(while (> @i 0)
  (println @i)
  (dosync (ref-set i (quot @i 2))))
```

2 ways without mutability:

```
(loop [i 1024]
  (when (pos? i)
    (println i)
    (recur (quot i 2))))


(doseq [i (take-while pos? (iterate #(quot % 2) 1024))]
  (println i))
```

COBOL (/wiki/Category:COBOL)

COBOL does not have a while loop construct, but it is does have a `PERFORM UNTIL` structure, which means that the normal condition used in a while loop must be negated.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Loop-While.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  I PIC 9999 VALUE 1024.

       PROCEDURE DIVISION.
           PERFORM UNTIL NOT 0 < I
               DISPLAY I
               DIVIDE 2 INTO I
           END-PERFORM

           GOBACK
           .
```

ColdFusion (/wiki/Category:ColdFusion)

Remove the leading space from the line break tag.

With tags:

```
<cfset i = 1024 /><cfloop condition="i GT 0">  #i#< br />
  <cfset i /= 2 />
</cfloop>
```

With script:

```
<cfscript>  i = 1024;
  while( i > 0 )
  {
    writeOutput( i + "< br/ >" );
  }
</cfscript>
```

Common Lisp (/wiki/Category:Common_Lisp)

```
(let ((i 1024))
  (loop while (plusp i) do
        (print i)
        (setf i (floor i 2))))

(loop with i = 1024
      while (plusp i) do
      (print i)
      (setf i (floor i 2)))

(defparameter *i* 1024)
(loop while (plusp *i*) do
      (print *i*)
      (setf *i* (floor *i* 2)))
```

Cowgol (/wiki/Category:Cowgol)

```
include "cowgol.coh";

var n: uint16 := 1024;
while n > 0 loop
    print_i16(n);
    print_nl();
    n := n/2;
end loop;
```

Crack (/wiki/Category:Crack)

```
i = 1024;
while( i > 0 ) {
  cout ` $i\n`;
  i = i/2;
}
```

Creative Basic (/mw/index.php?title=Category:Creative_Basic&action=edit&redlink=1)

```
DEF X:INT

X=1024

OPENCONSOLE

WHILE X>0

   PRINT X
   X=X/2

ENDWHILE
'Output starts with 1024 and ends with 1.

'Putting the following in the loop will produce output starting with 512 and ending with 0:
'X=X/2
'PRINT X

PRINT:PRINT"Press any key to end."

'Keep console from closing right away so the figures can be read.
WHILE INKEY$="":ENDWHILE

CLOSECONSOLE

'Since this is, in fact, a Creative Basic console program.
END
```

Note: Spacing is not an issue. I just find the code to be more readable with spaces.

Crystal (/wiki/Category:Crystal)

```
i = 1024
while i > 0
    puts i
    i //= 2
end
```

`until condition` is the negated version, equivalent to `while !(condition)`.

```
i = 1024
until i <= 0
   puts i
   i //= 2
end
```

D (/wiki/Category:D)

```
import std.stdio;

void main() {
    int i = 1024;

    while (i > 0) {
        writeln(i);
        i >>= 1;
    }
}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Dao (/wiki/Category:Dao)

```
i = 1024;
while( i > 0 ) i = i / 2;
```

Dc (/wiki/Category:Dc)

```
[ q ] sQ [ d 0!<Q p 2 / lW x ] sW 1024 lW x
```

DCL (/wiki/Category:DCL)

DCL is quite primitive in terms of "control statements", no WHILE, REPEAT, UNLESS or FOR, so must make do with IF/THEN/ELSE and GOTO statements.

```
$ i = 1024
$Loop:
$ IF ( i .LE. 0 ) THEN GOTO LoopEnd
$ WRITE sys$output F$FAO( "  i = !4UL", i )  ! formatted ASCII output, fixed-width field
$ ! Output alternatives:
$ !   WRITE sys$output F$STRING( i )         ! explicit integer-to-string conversion
$ !   WRITE sys$output i                     ! implicit conversion to string/output
$ i = i / 2
$ GOTO Loop
$LoopEnd:
```

Delphi (/wiki/Category:Delphi)

```
var
  i : Integer;
begin
  i := 1024;

  while i > 0 do
  begin
    Writeln(i);
    i := i div 2;
  end;
end;
```

Dragon (/wiki/Category:Dragon)

```
i = 1024
while(i > 0){
   showln i
   i >>= 1 //also acceptable: i /= 2
}
```

DUP (/wiki/Category:DUP)

```
1024[$][$.10,2/\%]# {Short form}
```

Explanation:

```
1024                {push 1024 on stack}
    [ ][         ]# {while[condition>0][do]}
     $              {DUP}
        $.          {DUP, print top of stack to STDOUT}
          10,       {print newline}
             2/\%   {2 DIV/MOD SWAP POP}
```

Alternative, if the interpreter allows using the shift operator:

```
1024[$][$.10,1»]#
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

DWScript (/wiki/Category:DWScript)

```
var i := 1024;

while i > 0 do begin
   PrintLn(i);
   i := i div 2;
end;
```

Dyalect (/wiki/Category:Dyalect)

Translation of: Swift

```
var i = 1024
while i > 0 {
  print(i)
  i /= 2
}
```

E (/wiki/Category:E)

```
var (http://wiki.erights.org/wiki/var) i := 1024
while (http://wiki.erights.org/wiki/while) (i > 0) {
    println (http://wiki.erights.org/wiki/println)(i)
    i //= 2
}
```

EasyLang (/wiki/Category:EasyLang)

```
i = 1024
while i > 0
  print i
  i = i div 2
.
```

EchoLisp (/wiki/Category:EchoLisp)

```

(set! n 1024)
(while (> n 0) (write n) (set! n (quotient n 2)))
1024 512 256 128 64 32 16 8 4 2 1

```

EGL (/wiki/Category:EGL)

```
x int = 1024;
while ( x > 0 )
   SysLib.writeStdout( x );
   x = MathLib.floor( x / 2 );
end
```

Elena (/wiki/Category:Elena)

ELENA 4.x:

```
public program()
{
    int i := 1024;
    while (i > 0)
    {
        console.writeLine:i;

        i /= 2
    }
}
```

Elixir (/wiki/Category:Elixir)

```
defmodule Loops do
  def while(0), do: :ok
  def while(n) do
    IO.puts n
    while( div(n,2) )
  end
end

Loops.while(1024)
```

Emacs Lisp (/wiki/Category:Emacs_Lisp)

```
(let ((i 1024))
  (while (> i 0)
    (message "%d" i)
    (setq i (/ i 2))))
```

Erlang (/wiki/Category:Erlang)

```
-module(while).
-export([loop/0]).

loop() ->
        loop(1024).

loop(N) when N div 2 =:= 0 ->
        io (http://erlang.org/doc/man/io.html):format("~w~n", [N]);

loop(N) when N >0 ->
        io (http://erlang.org/doc/man/io.html):format("~w~n", [N]),
        loop(N div 2).
```

ERRE (/wiki/Category:ERRE)

```

   I%=1024
   WHILE I%>0 DO  ! you can leave out >0
     PRINT(I%)
     I%=I% DIV 2  ! I%=INT(I%/2) for C-64 version
   END WHILE

```

Euphoria (/wiki/Category:Euphoria)

```
integer i
i = 1024

while i > 0 do
    printf(1, "%g\n", {i})
    i = floor(i/2) --Euphoria does NOT use integer division.  1/2 = 0.5
end while
```

Even without the `floor()` the code will in fact end. But it's FAR beyond 1.

F# (/wiki/Category:F_Sharp)

```
let rec loop n = if n > 0 then printf "%d " n; loop (n / 2)
loop 1024
```

Factor (/wiki/Category:Factor)

```
1024 [ dup 0 > ] [ dup . 2 /i ] while drop
```

FALSE (/wiki/Category:FALSE)

```
1024[$0>][$."
"2/]#%
```

Fantom (/wiki/Category:Fantom)

```
class Main
{
  public static Void main ()
  {
    Int i := 1024
    while (i > 0)
    {
      echo (i)
      i /= 2
    }
  }
}
```

Forth (/wiki/Category:Forth)

```
: halving ( n -- )
  begin  dup 0 >
  while  cr dup .  2/
  repeat drop ;
1024 halving
```

Fortran (/wiki/Category:Fortran)

Works with: Fortran (/wiki/Fortran) version 90 and later

```
INTEGER :: i = 1024
DO WHILE (i > 0)
  WRITE(*,*) i
  i = i / 2
END DO
```

Works with: Fortran (/wiki/Fortran) version 77 and later

```
      PROGRAM LOOPWHILE
        INTEGER I

C       FORTRAN 77 does not have a while loop, so we use GOTO statements
C       with conditions instead. This is one of two easy ways to do it.
        I = 1024
   10   CONTINUE
C       Check condition.
        IF (I .GT. 0) THEN
C         Handle I.
          WRITE (*,*) I
          I = I / 2
C         Jump back to before the IF block.
          GOTO 10
        ENDIF
        STOP
      END
```

Works with: Fortran (/wiki/Fortran) version IV and 66 and later

```
      PROGRAM LOOPWHILE
      INTEGER I
C     FORTRAN 66 does not have IF block.
      I = 1024
 10   CONTINUE
      IF (I .LE. 0) GOTO 20
      WRITE (*,*) I
      I = I / 2
      GOTO 10
 20   CONTINUE
      STOP
      END
```

Fortress (/wiki/Category:Fortress)

```
component loops_while
  export Executable

  var i:ZZ32 = 1024
  run() = while i > 0 do
    println(i)
    i := i DIV 2
  end
end
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

FreeBASIC (/wiki/Category:FreeBASIC)

```
' FB 1.05.0 Win64

Dim i As Integer = 1024

While i > 0
  Print i
  i Shr= 1
Wend

Sleep
```

Output:

```
 1024
 512
 256
 128
 64
 32
 16
 8
 4
 2
 1
```

Frink (/wiki/Category:Frink)

```
i=1024
while i>0
{
   i = i/1
}
```

FutureBasic (/wiki/Category:FutureBasic)

```

include "ConsoleWindow"

dim as long i : i = 1024

while i > 0
print i
i = int( i / 2 )
wend

```

Output:

```
 1024
 512
 256
 128
 64
 32
 16
 8
 4
 2
 1
```

Gambas (/wiki/Category:Gambas)

Click this link to run this code (https://gambas-playground.proko.eu/?gist=4e992013e4e7dc69a82477299a5ce23a)

```
Public (http://gambasdoc.org/help/lang/public) Sub (http://gambasdoc.org/help/lang/sub) Main()
Dim (http://gambasdoc.org/help/lang/dim) siCount As (http://gambasdoc.org/help/lang/as) Short (http://gambasdoc.org/help/lang/type/sho
rt) = 1024

While (http://gambasdoc.org/help/lang/while) siCount > 0
  Print (http://gambasdoc.org/help/lang/print) siCount;;
  siCount /= 2
Wend (http://gambasdoc.org/help/lang/wend)

End (http://gambasdoc.org/help/lang/end)
```

Output:

```
1024 512 256 128 64 32 16 8 4 2 1
```

GAP (/wiki/Category:GAP)

```
n := 1024;
while n > 0 do
    Print(n, "\n");
    n := QuoInt(n, 2);
od;
```

GML (/wiki/Category:GML)

```
i = 1024
while(i > 0)
    {
    show_message(string(i))
    i /= 2
    }
```

Go (/wiki/Category:Go)

```
i := 1024
for i > 0 {
  fmt.Printf("%d\n", i)
  i /= 2
}
```

Groovy (/wiki/Category:Groovy)

Solution:

```
int (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20int) i = 1024
while (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20while) (i > 0) {
    println (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20println) i
    i /= 2
}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Haskell (/wiki/Category:Haskell)

```
import Control.Monad (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Monad) (when)

main = loop 1024
  where loop n = when (n > 0)
                      (do print (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:print) n
                          loop (n `div (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:div)` 2))
```

You can use whileM_ function from monad-loops package that operates on monads:

```
import Data.IORef
import Control.Monad (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Monad).Loops

main :: IO (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:IO) ()
main = do r <- newIORef 1024
          whileM_ (do n <- readIORef r
                      return (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:return) (n > 0))
                  (do n <- readIORef r
                      print (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:print) n
                      modifyIORef r (`div (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:div)` 2))
```

With MonadComprehensions extension you can write it a little bit more readable:

```
{-# LANGUAGE MonadComprehensions #-}
import Data.IORef
import Control.Monad (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Monad).Loops

main :: IO (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:IO) ()
main = do
   r <- newIORef 1024
   whileM_ [n > 0 | n <- readIORef r] $ do
        n <- readIORef r
        print (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:print) n
        modifyIORef r (`div (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:div)` 2)
```

Haxe (/wiki/Category:Haxe)

Using shift right.

```
var i = 1024;

while (i > 0) {
  Sys.println(i);
  i >>= 1;
}
```

Using integer division.

```
var i = 1024;

while (i > 0) {
  Sys.println(i);
  i = Std.int(i / 2);
}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

hexiscript (/wiki/Category:Hexiscript)

```
let i 1024
while i > 0
  println i
  let i (i / 2)
endwhile
```

HolyC (/wiki/Category:HolyC)

```
U16 i = 1024;
while (i > 0) {
  Print("%d\n", i);
  i /= 2;
}
```

Icon (/wiki/Category:Icon) and Unicon (/wiki/Category:Unicon)

```
procedure main()
   local i
   i := 1024
   while write(0 < (i := i / 2))
end
```

Inform 7 (/wiki/Category:Inform_7)

```
let N be 1024;
while N > 0:
        say "[N][line break]";
        let N be N / 2;
```

IWBASIC (/wiki/Category:IWBASIC)

```

DEF X:INT

X=1024

OPENCONSOLE

WHILE X>0

    PRINT X
    X=X/2

ENDWHILE
'Output starts with 1024 and ends with 1.

'Putting the following in the loop will produce output starting with 512 and ending with 0:
'X=X/2
'PRINT X

'When compiled as a console only program, a press any key to continue message is automatic.
'I presume code is added by the compiler.
CLOSECONSOLE

'Since this is, in fact, an IWBASIC console program, which compiles and runs.
END
```

Note: Spacing is not an issue. I just find the code to be more readable with spaces.

J (/wiki/Category:J)

J is array-oriented, so there is very little need for loops. For example, one could satisfy this task this way:

```
,. <.@-:^:*^:a: 1024
```

J does support loops for those times they can't be avoided (just like many languages support gotos for those time they can't be avoided).

```
monad define 1024
  while. 0 < y do.
    smoutput y
    y =. <. -: y
  end.
  i.0 0
)
```

Note: this defines an anonymous function (monad define, and the subsequent lines) and passes it the argument 1024, which means it will be executed as soon as the full definition is available.

Java (/wiki/Category:Java)

```
int i = 1024;
while(i > 0){
   System (http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/System.html).out.println(i);
   i >>= 1; //also acceptable: i /= 2;
}
```

With a for loop:

```
for(int i = 1024; i > 0;i /= 2 /*or i>>= 1*/){
   System (http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/System.html).out.println(i);
}
```

JavaScript (/wiki/Category:JavaScript)

```
var n = 1024;
while (n > 0) {
  print(n);
  n /= 2;
}
```

In a functional idiom of JavaScript, however, we can not use a While **statement** to achieve this task, as statements return no value, mutate state, and can not be composed within other functional expressions.

Instead, we can define a composable loopWhile() **function** which has no side effects, and takes 3 arguments:

1. An initial value
2. A function which returns some derived value, corresponding to the body of the While loop
3. A conditional function, corresponding to the While test

```
function loopWhile(varValue, fnDelta, fnTest) {
  'use strict';
  var d = fnDelta(varValue);

  return fnTest(d) ? [d].concat(
    loopWhile(d, fnDelta, fnTest)
  ) : [];
}

console.log(
  loopWhile(
    1024,
    function (x) {
      return Math.floor(x/2);
    },
    function (x) {
      return x > 0;
    }
  ).join('\n')
);
```

If we assume integer division here (Math.floor(x/2)) rather than the floating point division (x/2) used in the imperative example, we obtain the output:

```
512
256
128
64
32
16
8
4
2
1
```

Joy (/wiki/Category:Joy)

```
DEFINE putln == put '\n putch.

1024 [] [dup putln 2 /] while.
```

jq (/wiki/Category:Jq)

Using recurse/1

```
# To avoid printing 0, test if the input is greater than 1
1024 | recurse( if . > 1 then ./2 | floor else empty end)
```

Using recurse/2 (requires jq >1.4)

```
1024 | recurse( ./2 | floor; . > 0)
```

Using a filter

```
def task: if . > 0 then ., (./2 | floor | task) else empty end;
1024|task
```

Using while/2

If your jq does not include while/2 as a builtin, here is its definition:

```
def while(cond; update):
  def _while: if cond then ., (update | _while) else empty end;
  _while;
```

For example:

```
1024|while(. > 0; ./2|floor)
```

Jsish (/wiki/Category:Jsish)

```
#!/usr/bin/env jsish
/* Loops/While in Jsish */
var i = 1024;

while (i > 0) { puts(i); i = i / 2 | 0; }

/*
=!EXPECTSTART!=
1024
512
256
128
64
32
16
8
4
2
1
=!EXPECTEND!=
*/
```

Output:

```
prompt$ jsish -u loopsWhile.jsi
[PASS] loopsWhile.jsi
```

Julia (/wiki/Category:Julia)

```

n = 1024

while n > 0
    println(n)
    n >>= 1
end

```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

K (/wiki/Category:K)

Implementation of the task using anonymous function is given below

```

{while[x>0; \echo x; x%:2]} 1024

```

Kotlin (/wiki/Category:Kotlin)

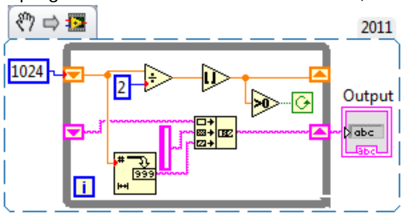
```
// version 1.0.6

fun main(args: Array<String>) {
    var (https://scala-lang.org) value = 1024
    while (https://scala-lang.org) (value > 0) {
        println(value)
        value /= 2
    }
}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

LabVIEW (/wiki/Category:LabVIEW)

Use Round Towards -Inf to prevent the integer becoming a float.
This image is a VI Snippet (http://zone.ni.com/devzone/cda/tut/p/id/9330), an executable image of LabVIEW (/wiki/LabVIEW) code. The LabVIEW version is shown on the top-right hand corner. You can download it, then drag-and-drop it onto the LabVIEW block diagram from a file browser, and it will appear as runnable, editable code.



(/wiki/File:LabVIEW_Loops_While.png)

Lambdatalk (/wiki/Category:Lambdatalk)

```
{def loops_while
 {lambda {:i}
  {if {< :i 1}
   then (end of loop)
   else :i {loops_while {/ :i 2}}}}}
-> loops_while

{loops_while 1024}
-> 1024 512 256 128 64 32 16 8 4 2 1 (end of loop)
```

Lang5 (/wiki/Category:Lang5)

Translation of: Factor

```
: /i  / int ; : 0=  0 == ;
: dip  swap '_ set execute _ ; : dupd  'dup dip ;
: 2dip  swap '_x set swap '_y set execute _y _x ;
: while
    do  dupd 'execute 2dip
        rot 0= if break else dup 2dip then
    loop ;

1024 "dup 0 >" "dup . 2 /i" while
```

Lambdatalk (/wiki/Category:Lambdatalk)

```
{def while
 {lambda {:i}
  {if {< :i 1}
   then
   else {br}:i {while {/ :i 2}}}}}

{while 1024} ->

1024
512
256
128
64
32
16
8
4
2
1
```

langur (/wiki/Category:Langur)

0.8 changed the keyword for a test only loop from for to while.

Works with: langur (/wiki/Langur) version 0.8

```
var .i = 1024
while .i > 0 {
    writeln .i
    .i \= 2
}
```

Works with: langur (/wiki/Langur) version < 0.8

```
var .i = 1024
for .i > 0 {
    writeln .i
    .i \= 2
}
```

Lasso (/wiki/Category:Lasso)

```
local(i = 1024)
while(#i > 0) => {^
        #i + '\r'
        #i /= 2
^}
```

Liberty BASIC (/wiki/Category:Liberty_BASIC)

All integers are changed to floats if an operation creates a non-integer result. Without using int() the program keeps going until erroring because accuracy was lost.

```
i = 1024
while i > 0
   print i
   i = int( i / 2)
wend
end
```

LIL (/wiki/Category:LIL)

```
set num 1024; while {$num > 0} {print $num; set num [expr $num \ 2]}
```

Backslash is integer division, otherwise LIL would allow the division to go floating point.

Lingo (/wiki/Category:Lingo)

```
n = 1024
repeat while n>0
  put n
  n = n/2 -- integer division implicitely returns floor: 1/2 -> 0
end repeat
```

Lisaac (/wiki/Category:Lisaac)

```
+ i : INTEGER;
i := 1024;
{ i > 0 }.while_do {
  i.println;

  i := i / 2;
};
```

LiveCode (/wiki/Category:LiveCode)

```
put 1024 into n
repeat while n > 0
    put n & cr
    divide n by 2
end repeat
```

Logo (/wiki/Category:Logo)

```
make "n 1024
while [:n > 0] [print :n  make "n :n / 2]
```

LOLCODE (/wiki/Category:LOLCODE)

LOLCODE's loop semantics require an afterthought if a condition is used, thus the `nop` in the following example. The more idiomatic approach would have been to `GTFO` of the loop once n had reached 0.

```
HAI 1.3

I HAS A n ITZ 1024

IM IN YR LOOP UPPIN YR nop WILE n
    VISIBLE n
    n R QUOSHUNT OF n AN 2
IM OUTTA YR LOOP

KTHXBYE
```

Lua (/wiki/Category:Lua)

```
n = 1024
while n>0 do
  print(n)
  n = math.floor(n/2)
end
```

M2000 Interpreter (/wiki/Category:M2000_Interpreter)

```

Module Checkit {
      Def long A=1024
      While A>0 {
            Print A
            A/=2
      }
}
Checkit

```

One line

```
Module Online { A=1024&: While A>0 {Print A: A/=2}} : OnLine
```

Make (/wiki/Category:Make)

```
NEXT=`expr $* / 2`
MAX=10

all: $(MAX)-n;

0-n:;

%-n: %-echo
       @-make -f while.mk $(NEXT)-n MAX=$(MAX)

%-echo:
       @echo $*
```

Invoking it

```
|make -f while.mk MAX=1024
```

Maple (/wiki/Category:Maple)

To avoid generating an infinite sequence (1/2, 1/4, 1/8, 1/16, etc.) of fractions after n takes the value 1, we use integer division (iquo) rather than the solidus operation (/).

```
> n := 1024: while n > 0 do print(n); n := iquo(n,2) end:
                                 1024
                                 512
                                 256
                                 128
                                  64
                                  32
                                  16
                                  8
                                  4
                                  2
                                  1
```

Mathematica (/wiki/Category:Mathematica)/Wolfram Language (/wiki/Category:Wolfram_Language)

Mathematica does not support integer-rounding, it would result in getting fractions: 1/2, 1/4 , 1/8 and so on; the loop would take infinite time without using the Floor functior

```
i = 1024;
While[i > 0,
 Print[i];
 i = Floor[i/2];
]
```

MATLAB (/wiki/Category:MATLAB) / Octave (/wiki/Category:Octave)

In Matlab (like Octave) the math is done floating point, then rounding to integer, so that 1/2 will be always 1 and never 0. A 'floor' is used to round the number.

```
i (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/i.html) = 1024;
while (i (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/i.html) > 0)
    disp (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/disp.html)(i (https://www.mathworks.com/access/helpdesk/help/tech
doc/ref/i.html));
    i (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/i.html) = floor (https://www.mathworks.com/access/helpdesk/help/tech
doc/ref/floor.html)(i (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/i.html)/2);
end
```

A vectorized version of the code is

```
  printf('%d\n', 2.^[log2 (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/log2.html)(1024):-1:0]);
```

Maxima (/wiki/Category:Maxima)

```
block([n], n: 1024, while n > 0 do (print(n), n: quotient(n, 2)));

/* using a C-like loop: divide control variable by two instead of incrementing it */
for n: 1024 next quotient(n, 2) while n > 0 do print(n);
```

MAXScript (/wiki/Category:MAXScript)

```
a = 1024
while a > 0 do
(
    print a
    a /= 2
)
```

Metafont (/wiki/Category:Metafont)

Metafont has no `while` loop, but it can be "simulated" easily.

```
a := 1024;
forever: exitif not (a > 0);
  show a;
  a := a div 2;
endfor
```

Microsoft Small Basic (/wiki/Category:Microsoft_Small_Basic)

```

i = 1024
While i > 0
  TextWindow.WriteLine(i)
  i = Math.Floor(i / 2)
EndWhile

```

min (/wiki/Category:Min)

Works with: min (/wiki/Min) version 0.19.3

```
1024 :n (n 0 >) (n puts 2 div @n) while
```

MiniScript (/wiki/Category:MiniScript)

```
i = 1024
while i > 0
    print i
    i = floor(i/2)
end while
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

MIRC Scripting Language (/wiki/Category:MIRC_Scripting_Language)

```
alias while_loop {
  var %n = 10
  while (%n >= 0) {
    echo (http://www.mirc.com/echo) -a Countdown: %n
    dec %n
  }
}
```

MIXAL (/mw/index.php?title=Category:MIXAL&action=edit&redlink=1)

```

******************************************
* X = M / N WHILE X > 0
* STORE EACH X IN NUMERIC ARRAY
* PRINT ARRAY
*******************************************
M       EQU     1024
N       EQU     2
LPR     EQU     18
BUF0    EQU     100
MSG     EQU     2000
LENGTH  EQU     500
        ORIG    3000
START   IOC     0(LPR)
        ENTX    M
CALC    STX     BUF0,1
        DIV     =N=
        SRAX    5
        INC1    1
        JXP     CALC
        ST1     LENGTH
PRINT   LDA     BUF0,2
        CHAR
        STX     MSG
        OUT     MSG(LPR)
        INC2    1
        CMP2    LENGTH
        JNE     PRINT
        HLT
        END     START

```

MK-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52)

```
1       0       2       4       П0      ИП0     /-/     x<0     15      ИП0
2       /       П0      БП      05      С/П
```

Modula-2 (/wiki/Category:Modula-2)

```
MODULE DivBy2;
  IMPORT InOut;

  VAR
    i: INTEGER;
BEGIN
  i := 1024;
  WHILE i > 0 DO
    InOut.WriteInt(i, 4);
    InOut.WriteLn;
    i := i DIV 2
  END
END DivBy2.
```

Modula-3 (/wiki/Category:Modula-3)

The usual module code and imports are omitted.

```
PROCEDURE DivBy2() =
  VAR i: INTEGER := 1024;
  BEGIN
    WHILE i > 0 DO
      IO.PutInt(i);
      IO.Put("\n");
      i := i DIV 2;
    END;
  END DivBy2;
```

Monte (/wiki/Category:Monte)

```
var i := 1024
while (i > 0):
    traceln(i)
    i //= 2
```

MOO (/wiki/Category:MOO)

```
i = 1024;
while (i > 0)
  player:tell(i);
  i /= 2;
endwhile
```

Morfa (/wiki/Category:Morfa)

```
import morfa.io.print;

var i = 1024;
while(i > 0)
{
    println(i);
    i /= 2;
}
```

Nanoquery (/wiki/Category:Nanoquery)

```
$n = 1024
while ($n > 0)
    println $n
    $n = $n/2
end while
```

Neko (/wiki/Category:Neko)

```
var i = 1024

while(i > 0) {
    $print(i + "\n");
    i = $idiv(i, 2)
}
```

Nemerle (/wiki/Category:Nemerle)

```
mutable x = 1024;
while (x > 0)
{
    WriteLine($"$x");
    x /= 2;
}
```

Or, with immutable types, after Haskell:

```
// within another function, eg Main()
def loop(n : int) : void
{
    when (n > 0)
    {
        WriteLine($"$n");
        loop(n / 2);
    }
}

loop(1024)
```

NetRexx (/wiki/Category:NetRexx)

```
/* NetRexx */
options replace format comments java crossref savelog symbols nobinary

  say
  say 'Loops/While'

  x_ = 1024
  loop while x_ > 0
    say x_.right(6)
    x_ = x_ % 2 -- integer division
    end
```

NewLISP (/wiki/Category:NewLISP)

```
(let (http://www.newlisp.org/downloads/newlisp_manual.html#let) (i 1024)
  (while (http://www.newlisp.org/downloads/newlisp_manual.html#while) (> i 0)
    (println (http://www.newlisp.org/downloads/newlisp_manual.html#println) i)
    (setq (http://www.newlisp.org/downloads/newlisp_manual.html#setq) i (/ i 2))))
```

Nim (/wiki/Category:Nim)

```
var n: int = 1024
while n > 0:
  echo(n)
  n = n div 2
```

NS-HUBASIC (/wiki/Category:NS-HUBASIC)

```
10 I=1024
20 IF I=0 THEN END
30 PRINT I
40 I=I/2
50 GOTO 20
```

Oberon-2 (/wiki/Category:Oberon-2)

The usual module code and imports are ommited.

```
PROCEDURE DivBy2*();
  VAR i: INTEGER;
BEGIN
  i := 1024;
  WHILE i > 0 DO
    Out.Int(i,0);
    Out.Ln;
    i := i DIV 2;
  END;
END DivBy2;
```

Objeck (/wiki/Category:Objeck)

```
i := 1024;
while(i > 0) {
   i->PrintLine();
   i /= 2;
};
```

OCaml (/wiki/Category:OCaml)

```
let n = ref 1024;;
while !n > 0 do
  Printf (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Printf.html).printf "%d\n" !n;
  n := !n / 2
done;;
```

But it is more common to write it in a tail-recursive functional style:

```
let rec loop n =
  if n > 0 then begin
    Printf (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Printf.html).printf "%d\n" n;
    loop (n / 2)
  end
in loop 1024
```

Octave (/wiki/Category:Octave)

```
i (http://octave.sourceforge.net/octave/function/i.html) = 1024;
while (i (http://octave.sourceforge.net/octave/function/i.html) > 0)
  disp (http://octave.sourceforge.net/octave/function/disp.html)(i (http://octave.sourceforge.net/octave/function/i.html))
  i (http://octave.sourceforge.net/octave/function/i.html) = floor (http://octave.sourceforge.net/octave/function/floor.html)(i (http:
//octave.sourceforge.net/octave/function/i.html)/2);
endwhile
```

The usage of the type int32 is not convenient, since the math is done floating point, then rounding to integer, so that 1/2 will be always 1 and never 0.

Oforth (/wiki/Category:Oforth)

```
1024 while ( dup ) [ dup println 2 / ]
```

OOC (/wiki/Category:OOC)

```

main: func {
  value := 1024
  while (value > 0) {
    value toString() println()
    value /= 2
  }
}

```

Oz (/wiki/Category:Oz)

Oz' for-loop can be used in a C-like manner:

```
for I in 1024; I>0; I div 2 do
   {Show I}
end
```

Alternatively, we can use the `while` feature of the for-loop with a mutable variable:

```
declare
  I = {NewCell 1024}
in
  for while:@I > 0 do
     {Show @I}
     I := @I div 2
  end
```

Panda (/wiki/Category:Panda)

Panda doesn't have explicit loops, instead we solve it by using the transitive closure operator. It applies a function to each successive value, each unique value is outputted. Our function halves, we make sure that the result is greater than 0 and add newline.

```
fun half(a) type integer->integer a.divide(2)
1024.trans(func:half).gt(0) nl
```

Panoramic (/wiki/Category:Panoramic)

```
dim x%:rem an integer

x%=1024

while x%>0

     print x%
     x%=x%/2

end_while

rem output starts with 1024 and ends with 1.

terminate
```

PARI/GP (/wiki/Category:PARI/GP)

```
n=1024;
while(n,
  print(n);
  n/=2
);
```

Pascal (/wiki/Category:Pascal)

```
program divby2(output);

var
  i: integer;

begin
  i := 1024;
  while i > 0 do
    begin
      writeln(i);
      i := i div 2
    end
end.
```

PeopleCode (/wiki/Category:PeopleCode)

```

Local string &CRLF;
Local number &LoopNumber;
&LoopNumber = 1024;
&CRLF = Char(10) | Char(13);

While &LoopNumber > 0;
 WinMessage(&LoopNumber | &CRLF);
 &LoopNumber = &LoopNumber / 2;
End-While;
```

Perl (/wiki/Category:Perl)

```
my $n = 1024;
while($n){
    print (https://perldoc.perl.org/functions/print.html) "$n\n";
    $n = int (https://perldoc.perl.org/functions/int.html) $n / 2;
}
```

or written as a for-loop and using the bit-shift operator

```
for(my $n = 1024; $n > 0; $n >>= 1){
    print (https://perldoc.perl.org/functions/print.html) "$n\n";
}
```

`until (condition)` is equivalent to `while (not condition)`.

```
my $n = 1024;
until($n == 0){
    print (https://perldoc.perl.org/functions/print.html) "$n\n";
    $n = int (https://perldoc.perl.org/functions/int.html) $n / 2;
}
```

Phix (/wiki/Category:Phix)

```
integer i = 1024
while i!=0 do
    ?i
    i = floor(i/2)  -- (see note)
end while
```

note: using i=i/2 would iterate over 1000 times until i is 4.94e-324 before the final division made it 0, if it didn't typecheck when it got set to 0.5

PHL (/wiki/Category:PHL)

```
var i = 1024;
while (i > 0) {
        printf("%i\n", i);
        i = i/2;
}
```

PHP (/wiki/Category:PHP)

```
$i = 1024;
while ($i > 0) {
   echo "$i\n";
   $i >>= 1;
}
```

PicoLisp (/wiki/Category:PicoLisp)

```
(let N 1024
   (while (gt0 N)
      (println N)
      (setq N (/ N 2)) ) )
```

Pike (/wiki/Category:Pike)

```
int main(){
   int i = 1024;
   while(i > 0){
      write(i + "\n");
      i = i / 2;
   }
}
```

PL/I (/wiki/Category:PL/I)

```
declare i fixed binary initial (1024);

do while (i>0);
   put skip list (i);
   i = i / 2;
end;
```

PL/SQL (/wiki/Category:PL/SQL)

Works with: Oracle (/wiki/Oracle)

```

SET (http://www.oracle.com/pls/db92/db92.drilldown?word=SET) serveroutput ON (http://www.oracle.com/pls/db92/db92.drilldown?word=ON)
DECLARE (http://www.oracle.com/pls/db92/db92.drilldown?word=DECLARE)
  n NUMBER (http://www.oracle.com/pls/db92/db92.drilldown?word=NUMBER) := 1024;
BEGIN (http://www.oracle.com/pls/db92/db92.drilldown?word=BEGIN)
  WHILE (http://www.oracle.com/pls/db92/db92.drilldown?word=WHILE) n > 0 LOOP (http://www.oracle.com/pls/db92/db92.drilldown?word=LOOP
)
    DBMS_OUTPUT (http://www.oracle.com/pls/db92/db92.drilldown?word=DBMS_OUTPUT).put_line(n);
    n := TRUNC (http://www.oracle.com/pls/db92/db92.drilldown?word=TRUNC)(n / 2);
  END (http://www.oracle.com/pls/db92/db92.drilldown?word=END) LOOP (http://www.oracle.com/pls/db92/db92.drilldown?word=LOOP);
END (http://www.oracle.com/pls/db92/db92.drilldown?word=END);
/

```

Plain English (/wiki/Category:Plain_English)

```
To run:
Start up.
Show the halvings of 1024.
Wait for the escape key.
Shut down.

To show the halvings of a number:
If the number is 0, exit.
Convert the number to a string.
Write the string to the console.
Divide the number by 2.
Repeat.
```

Pop11 (/wiki/Category:Pop11)

```
lvars i = 1024;
while i > 0 do
    printf(i, '%p\n');
    i div 2 -> i;
endwhile;
```

PostScript (/wiki/Category:PostScript)

PostScript has no real `while` loop, but it can easily be created with an endless loop and a check at the beginning:

```
1024
{
    dup 0 le     % check whether still greater than 0
    { pop exit } % if not, exit the loop
    if
    dup =        % print the number
    2 idiv       % divide by two
}
loop
```

PowerShell (/wiki/Category:PowerShell)

```
[int]$i = 1024
while ($i -gt 0) {
    $i
    $i /= 2
}
```

Prolog (/wiki/Category:Prolog)

```
while(0) :- !.
while(X) :-
    writeln(X),
    X1 is (http://pauillac.inria.fr/~deransar/prolog/bips.html) X // 2,
    while(X1).
```

Start the calculation at a top-level like this:

```
?- while(1024).
```

PureBasic (/wiki/Category:PureBasic)

```
If OpenConsole()

  x.i = 1024
  While x > 0
    PrintN(Str(x))
    x / 2
  Wend

  Print(#CRLF$ + #CRLF$ + "Press ENTER to exit")
  Input()
  CloseConsole()
EndIf
```

Python (/wiki/Category:Python)

```
n = 1024
while n > 0:
    print n
    n //= 2
```

QB64 (/wiki/Category:QB64)

```
Dim n As Integer
n = 1024
While n > 0
    Print n
    n = n \ 2
Wend
```

Quackery (/wiki/Category:Quackery)

```
1024
[ dup 0 > while
  dup echo cr 2 /
  again ]
drop
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

R (/wiki/Category:R)

```
i <- 1024L
while(i > 0)
{
   print(i)
   i <- i %/% 2
}
```

Racket (/wiki/Category:Racket)

Loop/When

```
#lang racket
(let loop ([n 1024])
  (when (positive? n)
    (displayln n)
    (loop (quotient n 2))))
```

Macro

```
#lang racket
(define-syntax-rule (while condition body ...)
  (let loop ()
    (when condition
      body ...
      (loop))))

(define n 1024)
(while (positive? n)
  (displayln n)
  (set! n (sub1 n)))
```

Raku (/wiki/Category:Raku)

(formerly Perl 6)

Here is a straightforward translation of the task description:

```
my $n = 1024; while $n > 0 { say $n; $n div= 2 }
```

The same thing with a C-style loop and a bitwise shift operator:

```
loop (my $n = 1024; $n > 0; $n +>= 1) { say $n }
```

And here's how you'd *really* write it, using a sequence operator that intuits the division for you:

```
.say for 1024, 512, 256 ... 1
```

REBOL (/wiki/Category:REBOL)

```
rebol [
        Title: "Loop/While"
        URL: http://rosettacode.org/wiki/Loop/While
]

value: 1024
while [value > 0][
        print value
        value: to-integer value / 2
]
```

Retro (/wiki/Category:Retro)

```
1024 [ cr &putn sip 2 / dup ] while
```

REXX (/wiki/Category:REXX)

version 1, simple

```
/*REXX program demonstrates a  DO WHILE  with index reduction construct.*/
j=1024                            /*define the initial value of  J.*/
        do  while  j>0                /*test if made at the top of  DO.*/
        say j
        j=j%2                         /*in REXX, % is integer division.*/
        end
                                  /*stick a fork in it, we're done.*/
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

version 2, right justified

Note that a faster version could be implemented with

DO WHILE x\==0

but that wouldn't be compliant with the wording of the task.

```
/*REXX program demonstrates a  DO WHILE  with index reduction construct.*/
x=1024                            /*define the initial value of  X.*/
        do  while  x>0                /*test if made at the top of  DO.*/
        say right(x,10)               /*pretty output by aligning right*/
        x=x%2                         /*in REXX, % is integer division.*/
        end
                                  /*stick a fork in it, we're done.*/
```

Output:

```
      1024
       512
       256
       128
        64
        32
        16
         8
         4
         2
         1
```

version 3, faster WHILE comparison

```
/*REXX program demonstrates a  DO WHILE  with index reduction construct.*/
x=1024                            /*define the initial value of  X.*/
        do  while  x>>0               /*this is an  exact  comparison. */
        say right(x,10)               /*pretty output by aligning right*/
        x=x%2                         /*in REXX, % is integer division.*/
        end
                                  /*stick a fork in it, we're done.*/
```

output is the same as version 2.

version 4, index reduction

```
/*REXX program demonstrates a  DO WHILE  with index reduction construct.*/
                                  /* [↓] note:   BY   defaults to 1*/
        do j=1024  by 0  while  j>>0  /*this is an  exact  comparison. */
        say right(j,10)               /*pretty output by aligning right*/
        j=j%2                         /*in REXX, % is integer division.*/
        end
                                  /*stick a fork in it, we're done.*/
```

output is the same as version 2.

Ring (/wiki/Category:Ring)

```
i = 1024
while i > 0
      see i + nl
      i = floor(i / 2)
end
```

Ruby (/wiki/Category:Ruby)

```
i = 1024
while i > 0 do
   puts i
   i /= 2
end
```

The above can be written in one statement:

```
puts i = 1024
puts i /= 2 while i > 0
```

`until condition` is equivalent to `while not condition`.

```
i = 1024
until i <= 0 do
   puts i
   i /= 2
end
```

Run BASIC (/wiki/Category:Run_BASIC)

```
i = 1024
while i > 0
   print i
   i = int(i / 2)
wend
end
```

Rust (/wiki/Category:Rust)

```
fn main() {
    let mut n: i32 = 1024;
    while n > 0 {
        println!("{}", n);
        n /= 2;
    }
}
```

SAS (/wiki/Category:SAS)

```
data _null_;
n=1024;
do while(n>0);
  put n;
  n=int(n/2);
end;
run;
```

Sather (/wiki/Category:Sather)

```
class MAIN is
  main is
    i ::= 1024;
    loop while!(i > 0);
      #OUT + i + "\n";
      i := i / 2;
    end;
  end;
end;
```

Scala (/wiki/Category:Scala)

Library: Scala (/wiki/Category:Scala)

Imperative

```
var (https://scala-lang.org) i = 1024
while (https://scala-lang.org) (i > 0) {
  println(i)
  i /= 2
}
```

Tail recursive

```
  @tailrec
  def (https://scala-lang.org) loop(iter: Int) {
    if (https://scala-lang.org) (iter > 0) {
      println(iter)
      loop(iter / 2)
    }
  }
  loop(1024)
```

Iterator

```
  def (https://scala-lang.org) loop = new (https://scala-lang.org) Iterator[Int] {
    var (https://scala-lang.org) i = 1024
    def (https://scala-lang.org) hasNext = i > 0
    def (https://scala-lang.org) next(): Int = { val (https://scala-lang.org) tmp = i; i = i / 2; tmp }
  }
  loop.foreach(println(_))
```

Stream

Finite stream (1024..0) filtered by takeWhile (1024..1).

```
  def (https://scala-lang.org) loop(i: Int): Stream[Int] = i #:: (if (https://scala-lang.org) (i > 0) loop(i / 2) else (https://scala-
lang.org) Stream.empty)
  loop(1024).takeWhile(_ > 0).foreach(println(_))
```

Scheme (/wiki/Category:Scheme)

```
(do ((n 1024 (quotient n 2)))
    ((<= n 0))
    (display n)
    (newline))
```

Scilab (/wiki/Category:Scilab)

Works with: Scilab (/wiki/Scilab) version 5.5.1

```
i=1024
while i>0
    printf("%4d\n",i)
    i=int(i/2)
end
```

Output:

```
1024
 512
 256
 128
  64
  32
  16
   8
   4
   2
   1
```

Seed7 (/wiki/Category:Seed7)

```
$ include "seed7_05.s7i";

const proc: main is func
  local
    var integer: i is 1024;
  begin
    while i > 0 do
      writeln(i);
      i := i div 2
    end while;
  end func;
```

SenseTalk (/wiki/Category:SenseTalk)

```
put 1024 into x
log x
Repeat until x = 1
        divide x by 2
        log x
End repeat
```

SETL (/wiki/Category:SETL)

```
n := 1024;
while n > 0 loop
    print( n );
    n := n div 2;
end loop;
```

Sidef (/wiki/Category:Sidef)

```
var i = 1024
while (i > 0) {
    say i
    i //= 2
}
```

Simula (/wiki/Category:Simula)

Works with: SIMULA-67 (/mw/index.php?title=SIMULA-67&action=edit&redlink=1)

```
begin
  integer i;
  i:=1024;
  while i>0 do
  begin
     outint(i,5);
     i:=i//2-1
  end
end
```

Output:

```
 1024  511  254  126   62   30   14    6    2
```

Sinclair ZX81 BASIC (/wiki/Category:Sinclair_ZX81_BASIC)

The distinctive thing about a `while` loop is that the conditional test happens before the loop body, not after—so that the code in the loop may be executed zero times.

Since we have no integer type, we floor the result of the division each time.

```
10 LET I=1024
20 IF I=0 THEN GOTO 60
30 PRINT I
40 LET I=INT (I/2)
50 GOTO 20
```

Slate (/wiki/Category:Slate)

```
#n := 1024.
[n isPositive] whileTrue:
  [inform: number printString.
   n := n // 2]
```

Smalltalk (/wiki/Category:Smalltalk)

The Block (aka lambda closure) class provides a number of loop messages; with test at begin, test at end and with exit (break).

```
[s atEnd] whileFalse: [s next. ...].
[foo notNil] whileTrue: [s next. ...].
[...] doWhile: [ ... someBooleanExpression ].
[...] doUntil: [ ... someBooleanExpression ].
```

[:exit | ... cold ifTrue:[exit value]. ...] loopWithExit</lang>

Examples:

```
number := 1024.
[ number > 0 ] whileTrue:
  [ Transcript print: number; nl.
  number := number // 2 ]
```

```
number := 1024.
[ number <= 0 ] whileFalse:
  [ Transcript print: number; nl.
  number := number // 2 ]
```

Sparkling (/wiki/Category:Sparkling)

```
var i = 1024;
while i > 0 {
    print(i);
    i /= 2;
}
```

Spin (/wiki/Category:Spin)

Works with: BST/BSTC (/mw/index.php?title=BST/BSTC&action=edit&redlink=1)
Works with: FastSpin/FlexSpin (/mw/index.php?title=FastSpin/FlexSpin&action=edit&redlink=1)
Works with: HomeSpun (/mw/index.php?title=HomeSpun&action=edit&redlink=1)
Works with: OpenSpin (/mw/index.php?title=OpenSpin&action=edit&redlink=1)

```
con
  _clkmode = xtal1 + pll16x
  _clkfreq = 80_000_000

obj
  ser : "FullDuplexSerial.spin"

pub main | n
  ser.start(31, 30, 0, 115200)

  n := 1024
  repeat while n > 0
    ser.dec(n)
    ser.tx(32)
    n /= 2

  waitcnt(_clkfreq + cnt)
  ser.stop
  cogstop(0)
```

Output:

```
1024 512 256 128 64 32 16 8 4 2 1
```

SPL (/wiki/Category:SPL)

```
n = 1024
>
  #.output(n)
  n /= 2
< n!<1
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

SQL PL (/wiki/Category:SQL_PL)

Works with: Db2 LUW (/wiki/Db2_LUW)
version 9.7 or higher.
With SQL PL:

```
--#SET TERMINATOR @

SET SERVEROUTPUT ON @

BEGIN
 DECLARE I SMALLINT DEFAULT 1024;

 Loop: WHILE (I > 0) DO
  CALL DBMS_OUTPUT.PUT_LINE(I);
  SET I = I / 2;
 END WHILE Loop;
END @
```

Output:

```
db2 -td@
db2 => SET SERVEROUTPUT ON @
DB20000I  The SET SERVEROUTPUT command completed successfully.
db2 => BEGIN
...
db2 (cont.) => END @
DB20000I  The SQL command completed successfully.

1024
512
256
128
64
32
16
8
4
2
1
```

Standard ML (/wiki/Category:Standard_ML)

```
val n = ref 1024;
while !n > 0 do (
  print (Int.toString (!n) ^ "\n");
  n := !n div 2
)
```

But it is more common to write it in a tail-recursive functional style:

```
let
  fun loop n =
    if n > 0 then (
      print (Int.toString n ^ "\n");
      loop (n div 2)
    ) else ()
in
  loop 1024
end
```

Stata (/wiki/Category:Stata)

```
local n=1024
while `n'>0 {
        display `n'
        local n=floor(`n'/2)
}
```

Suneido (/wiki/Category:Suneido)

```
i = 1024
while (i > 0)
    {
    Print(i)
    i = (i / 2).Floor()
    }
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Swift (/wiki/Category:Swift)

```
var i = 1024
while i > 0 {
  println(i)
  i /= 2
}
```

Tailspin (/wiki/Category:Tailspin)

In Tailspin you can loop by sending a value back to the matchers (by "-> #"). Depending on how you set that up, you create different loops.

```

1024 -> \(
  <0~..> '$;$#10;' -> !OUT::write
    $ ~/ 2 -> #
\) -> !VOID

```

Tcl (/wiki/Category:Tcl)

```
set i 1024
while {$i > 0} {
    puts $i
    set i [expr {$i / 2}]
}
```

Plain $\TeX$ (/wiki/Category:PlainTeX)

```
\newcount\rosetta
\rosetta=1024
\loop
    \the\rosetta\endgraf
    \divide\rosetta by 2
    \ifnum\rosetta > 0
\repeat
\end
```

TI-83 BASIC (/wiki/Category:TI-83_BASIC)

```
1024→I
While I>0
Disp I
I/2→I
End

```

TI-89 BASIC (/wiki/Category:TI-89_BASIC)

```
Local i
1024 → i
While i > 0
  Disp i
  intDiv(i, 2) → i
EndWhile
```

TorqueScript (/wiki/Category:TorqueScript)

This has to make use of mFloor because torque has automatic type shuffling, causing an infiniteloop.

```
%num = 1024;
while(%num > 0)
{
    echo(%num);
    %num = mFloor(%num / 2);
}
```

Transact-SQL (/wiki/Category:Transact-SQL)

```
DECLARE @i INT = 1024;
WHILE @i >0
BEGIN
    PRINT @i;
    SET @i = @i / 2;
END;
```

Trith (/wiki/Category:Trith)

```
1024 [dup print 2 / floor] [dup 0 >] while drop
```

```
1024 [dup print 1 shr] [dup 0 >] while drop
```

TUSCRIPT (/wiki/Category:TUSCRIPT)

```
$$ MODE TUSCRIPT
i=1024
LOOP
   PRINT i
   i=i/2
   IF (i==0) EXIT
ENDLOOP
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Unicon (/wiki/Category:Unicon)

See Icon.

Uniface (/wiki/Category:Uniface)

```
variables
        numeric I
endvariables

I = 1024
while (I > 0)
        putmess I
        I = (I/2)[trunc]
endwhile
```

UNIX Shell (/wiki/Category:UNIX_Shell)

Works with: Bourne Again SHell (/wiki/Bourne_Again_SHell)

```
x=1024
while [[ $x -gt 0 ]]; do
  echo $x
  x=$(( $x/2 ))
done
```

UnixPipes (/wiki/Category:UnixPipes)

```
(echo 1024>p.res;tail -f p.res) | while read a ; do
   test $a -gt 0 && (expr $a / 2  >> p.res ; echo $a) || exit 0
done
```

Ursa (/wiki/Category:Ursa)

```
decl int n
set n 1024

while (> n 0)
    out n endl console
    set n (int (/ n 2))
end while
```

Ursala (/wiki/Category:Ursala)

Unbounded iteration is expressed with the -> operator. An expression (p-> f) x, where p is a predicate and f is a function, evaluates to x, f(x), or f(f(x)), etc. as far as necessar to falsify p.

Printing an intermediate result on each iteration is a bigger problem because side effects are awkward. Instead, the function g in this example iteratively constructs a list of results, which is displayed on termination.

The argument to g is the unit list <1024>. The predicate p is ~&h, the function that tests whether the head of a list is non-null (equivalent to non-zero). The iterated function f is that which conses the truncated half of the head of its argument with a copy of the whole argument. The main program takes care of list reversal and formatting.

```
#import nat

g = ~&h-> ^C/half@h ~&

#show+

main = %nP*=tx g <1024>
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Explicit iteration has its uses but there are always alternatives. The same output is produced by the following main program using bit manipulation.

```
main = %nP*=tK33 1024
```

V (/wiki/Category:V)

```
1024 [0 >] [
   dup puts
   2 / >int
] while
```

Vala (/wiki/Category:Vala)

```
int i = 1024;
while (i > 0) {
  stdout.printf("%d\n", i);
  i /= 2;
}
```

VBA (/wiki/Category:VBA)

```
Public Sub LoopsWhile()
    Dim value As Integer
    value = 1024
    Do While value > 0
        Debug.Print value
        value = value / 2
    Loop
End Sub
```

Vedit macro language (/wiki/Category:Vedit_macro_language)

```
#1 = 1024
while (#1 > 0) {
    Num_Type(#1)
    #1 /= 2
}
```

or with for loop:

```
for (#1 = 1024; #1 > 0; #1 /= 2) {
    Num_Type(#1)
}
```

Verbexx (/wiki/Category:Verbexx)

```
//  Basic @LOOP while: verb

@LOOP init:{@VAR n = 1024} while:(n > 0) next:{n /= 2}
{
     @SAY n;
};
```

Verilog (/wiki/Category:Verilog)

```


module main;
  integer i;

  initial begin
      i = 1024;

      while( i > 0) begin
        $display(i);
        i = i / 2;
      end
      $finish ;
    end
endmodule

```

Vim Script (/wiki/Category:Vim_Script)

```
let i = 1024
while i > 0
    echo i
    let i = i / 2
endwhile
```

Visual Basic .NET (/wiki/Category:Visual_Basic_.NET)

```
Dim x = 1024
Do
    Console.WriteLine(x)
    x = x \ 2
Loop While x > 0
```

Wart (/wiki/Category:Wart)

```
i <- 1024
while (i > 0)
  prn i
  i <- (int i/2)
```

Wee Basic (/wiki/Category:Wee_Basic)

```
let number=1024
while number>0.5
print 1 number
let number=number/2
wend
end
```

Whitespace (/wiki/Category:Whitespace)

```

```

Pseudo-assembly equivalent:

```
push 1024

0:
    dup onum push 10 ochr
    push 2 div dup
    push 0 swap sub
        jn 0
        pop exit
```

Wren (/wiki/Category:Wren)

```
var i = 1024
while (i > 0) {
    System.print(i)
    i = (i / 2).floor
}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

X86 Assembly (/wiki/Category:X86_Assembly)

```
; NASM 64 bit X86-64 assembly on Linux

global main
extern printf

segment .data

printffmt db `%ld\n`,0

segment .text

main:
    push rbp
    mov rbp,rsp

; used rbx and r12 because printf preserves these values

    mov rbx,1024                ; start with 1024
    mov r12,2                   ; load 2 as divisor

.toploop                        ; top of while loop
    cmp rbx,0                   ; compare to 0
    jle .done                   ; exit 0 or less

    lea rdi,[printffmt]         ; print number in rsi
    mov rsi,rbx                 ; mov to rsi as argument
    call printf

; calculate n/2 and save
    xor rdx,rdx                 ; clear rdx for division
    mov rax,rbx                 ; mov number to rax for division
    idiv r12                    ; divide by 2
    mov rbx,rax                 ; save n/2

    jmp .toploop                ; next loop

.done
    xor rax,rax                 ; return code 0
    leave                       ; fix stack
    ret                         ; return
```

XBasic (/wiki/Category:XBasic)

Works with: Windows XBasic (/wiki/Windows_XBasic)

```
i% = 1024
DO WHILE i% > 0
  PRINT i%
  i% = i% / 2
LOOP
```

XLISP (/wiki/Category:XLISP)

The specification calls for an integer value and for the loop to run `WHILE` that value is greater than zero. In a dynamically typed language like XLISP, variables cannot be declared as integer or real; but the same result is obtained by looping `WHILE` the value of the variable *i* is greater than or equal to one.

```
(DEFINE I 1024)

(WHILE (>= I 1)
    (PRINT I)
    (DEFINE I (/ I 2)))
```

XPL0 (/wiki/Category:XPL0)

```
code CrLf=9, IntOut=11;
int I;
[I:= 1024;
while I>0 do
        [IntOut(0, I);  CrLf(0);
        I:= I>>1;       \(same as I/2 for positive I)
        ];
]
```

Yabasic (/wiki/Category:Yabasic)

```
i = 1024

while i > 0
  Print i
  i = int(i / 2)
wend

end
```

zkl (/wiki/Category:Zkl)

```
n:=1024; while(n>0){println(n); n/=2;}
```

Output:

```
1024
512
256
128
64
32
16
8
4
2
1
```

Categories (/wiki/Special:Categories): Programming Tasks (/wiki/Category:Programming_Tasks) | Iteration (/wiki/Category:Iteration) | Conditional loops (/wiki/Category:Conditional_loops) | Simple (/wiki/Category:Simple) | GUISS/Omit (/wiki/Category:GUISS/Omit) | 0815 (/wiki/Category:0815) | 11l (/wiki/Category:11l) | 360 Assembly (/wiki/Category:360_Assembly) | 6502 Assembly (/wiki/Category:6502_Assembly) | AArch64 Assembly (/wiki/Category:AArch64_Assembly) | ActionScript (/wiki/Category:ActionScript) | Ada (/wiki/Category:Ada) | Agena (/wiki/Category:Agena) | Aime (/wiki/Category:Aime) | ALGOL 60 (/wiki/Category:ALGOL_60) | ALGOL 68 (/wiki/Category:ALGOL_68) | ALGOL W (/wiki/Category:ALGOL_W) | ALGOL-M (/wiki/Category:ALGOL-M) | AmbientTalk (/wiki/Category:AmbientTalk) | AmigaE (/wiki/Category:AmigaE) | AppleScript (/wiki/Category:AppleScript) | Applesoft BASIC (/wiki/Category:Applesoft_BASIC) | ARM Assembly (/wiki/Category:ARM_Assembly) | ArnoldC (/wiki/Category:ArnoldC) | Arturo (/wiki/Category:Arturo) | AutoHotkey (/wiki/Category:AutoHotkey) | AWK (/wiki/Category:AWK) | Axe (/wiki/Category:Axe) | BASIC (/wiki/Category:BASIC) | BaCon (/wiki/Category:BaCon) | Commodore BASIC (/wiki/Category:Commodore_BASIC) | BBC BASIC (/wiki/Category:BBC_BASIC) | IS-BASIC (/wiki/Category:IS-BASIC) | BASIC256 (/wiki/Category:BASIC256) | Bc (/wiki/Category:Bc) | Befunge (/wiki/Category:Befunge) | Blz (/wiki/Category:Blz) | Bracmat (/wiki/Category:Bracmat) | Brat (/wiki/Category:Brat) | C (/wiki/Category:C) | C sharp (/wiki/Category:C_sharp) | C++ (/wiki/Category:C%2B%2B) | Caché ObjectScript (/wiki/Category:Cach%C3%A9_ObjectScript) | Chapel (/wiki/Category:Chapel) | ChucK (/wiki/Category:ChucK) | Clojure (/wiki/Category:Clojure) | COBOL (/wiki/Category:COBOL) | ColdFusion (/wiki/Category:ColdFusion) | Common Lisp (/wiki/Category:Common_Lisp) | Cowgol (/wiki/Category:Cowgol) | Crack (/wiki/Category:Crack) | Creative Basic (/mw/index.php?title=Category:Creative_Basic&action=edit&redlink=1) | Crystal (/wiki/Category:Crystal) | D (/wiki/Category:D) | Dao (/wiki/Category:Dao) | Dc (/wiki/Category:Dc) | DCL (/wiki/Category:DCL) | Delphi (/wiki/Category:Delphi) | Dragon (/wiki/Category:Dragon) | DUP (/wiki/Category:DUP) | DWScript (/wiki/Category:DWScript) | Dyalect (/wiki/Category:Dyalect) | E (/wiki/Category:E) | EasyLang (/wiki/Category:EasyLang) | EchoLisp (/wiki/Category:EchoLisp) | EGL (/wiki/Category:EGL) | Elena (/wiki/Category:Elena) | Elixir (/wiki/Category:Elixir) | Emacs Lisp (/wiki/Category:Emacs_Lisp) | Erlang (/wiki/Category:Erlang) | ERRE (/wiki/Category:ERRE) | Euphoria (/wiki/Category:Euphoria) | F Sharp (/wiki/Category:F_Sharp) | Factor (/wiki/Category:Factor) | FALSE (/wiki/Category:FALSE) | Fantom (/wiki/Category:Fantom) | Forth (/wiki/Category:Forth) | Fortran (/wiki/Category:Fortran) | Fortress (/wiki/Category:Fortress) | FreeBASIC (/wiki/Category:FreeBASIC) | Frink (/wiki/Category:Frink) | FutureBasic (/wiki/Category:FutureBasic) | Gambas (/wiki/Category:Gambas) | GAP (/wiki/Category:GAP) | GML (/wiki/Category:GML) | Go (/wiki/Category:Go) | Groovy (/wiki/Category:Groovy) | Haskell (/wiki/Category:Haskell) | Haxe (/wiki/Category:Haxe) | Hexiscript (/wiki/Category:Hexiscript) | HolyC (/wiki/Category:HolyC) | Icon (/wiki/Category:Icon) | Unicon (/wiki/Category:Unicon) | Inform 7 (/wiki/Category:Inform_7) | IWBASIC (/wiki/Category:IWBASIC) | J (/wiki/Category:J) | Java (/wiki/Category:Java) | JavaScript (/wiki/Category:JavaScript) | Joy (/wiki/Category:Joy) | Jq (/wiki/Category:Jq) | Jsish (/wiki/Category:Jsish) | Julia (/wiki/Category:Julia) | K (/wiki/Category:K) | Kotlin (/wiki/Category:Kotlin) | LabVIEW (/wiki/Category:LabVIEW) | Lambdatalk (/wiki/Category:Lambdatalk) | Lang5 (/wiki/Category:Lang5) | Langur (/wiki/Category:Langur) | Lasso (/wiki/Category:Lasso) | Liberty BASIC (/wiki/Category:Liberty_BASIC) | LIL (/wiki/Category:LIL) | Lingo (/wiki/Category:Lingo) | Lisaac (/wiki/Category:Lisaac) | LiveCode (/wiki/Category:LiveCode) | Logo (/wiki/Category:Logo) | LOLCODE (/wiki/Category:LOLCODE) | Lua (/wiki/Category:Lua)

| M2000 Interpreter (/wiki/Category:M2000_Interpreter) | Make (/wiki/Category:Make) | Maple (/wiki/Category:Maple) | Mathematica (/wiki/Category:Mathematica)
| Wolfram Language (/wiki/Category:Wolfram_Language) | MATLAB (/wiki/Category:MATLAB) | Octave (/wiki/Category:Octave) | Maxima (/wiki/Category:Maxima)
| MAXScript (/wiki/Category:MAXScript) | Metafont (/wiki/Category:Metafont) | Microsoft Small Basic (/wiki/Category:Microsoft_Small_Basic) | Min (/wiki/Category:Min)
| MiniScript (/wiki/Category:MiniScript) | MIRC Scripting Language (/wiki/Category:MIRC_Scripting_Language)
| MIXAL (/mw/index.php?title=Category:MIXAL&action=edit&redlink=1) | MK-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52) | Modula-2 (/wiki/Category:Modula-2)
| Modula-3 (/wiki/Category:Modula-3) | Monte (/wiki/Category:Monte) | MOO (/wiki/Category:MOO) | Morfa (/wiki/Category:Morfa) | Nanoquery (/wiki/Category:Nanoquery)
| Neko (/wiki/Category:Neko) | Nemerle (/wiki/Category:Nemerle) | NetRexx (/wiki/Category:NetRexx) | NewLISP (/wiki/Category:NewLISP) | Nim (/wiki/Category:Nim)
| NS-HUBASIC (/wiki/Category:NS-HUBASIC) | Oberon-2 (/wiki/Category:Oberon-2) | Objeck (/wiki/Category:Objeck) | OCaml (/wiki/Category:OCaml)
| Oforth (/wiki/Category:Oforth) | OOC (/wiki/Category:OOC) | Oz (/wiki/Category:Oz) | Panda (/wiki/Category:Panda) | Panoramic (/wiki/Category:Panoramic)
| PARI/GP (/wiki/Category:PARI/GP) | Pascal (/wiki/Category:Pascal) | PeopleCode (/wiki/Category:PeopleCode) | Perl (/wiki/Category:Perl) | Phix (/wiki/Category:Phix)
| PHL (/wiki/Category:PHL) | PHP (/wiki/Category:PHP) | PicoLisp (/wiki/Category:PicoLisp) | Pike (/wiki/Category:Pike) | PL/I (/wiki/Category:PL/I)
| PL/SQL (/wiki/Category:PL/SQL) | Plain English (/wiki/Category:Plain_English) | Pop11 (/wiki/Category:Pop11) | PostScript (/wiki/Category:PostScript)
| PowerShell (/wiki/Category:PowerShell) | Prolog (/wiki/Category:Prolog) | PureBasic (/wiki/Category:PureBasic) | Python (/wiki/Category:Python)
| QB64 (/wiki/Category:QB64) | Quackery (/wiki/Category:Quackery) | R (/wiki/Category:R) | Racket (/wiki/Category:Racket) | Raku (/wiki/Category:Raku)
| REBOL (/wiki/Category:REBOL) | Retro (/wiki/Category:Retro) | REXX (/wiki/Category:REXX) | Ring (/wiki/Category:Ring) | Ruby (/wiki/Category:Ruby)
| Run BASIC (/wiki/Category:Run_BASIC) | Rust (/wiki/Category:Rust) | SAS (/wiki/Category:SAS) | Sather (/wiki/Category:Sather) | Scala (/wiki/Category:Scala)
| Scheme (/wiki/Category:Scheme) | Scilab (/wiki/Category:Scilab) | Seed7 (/wiki/Category:Seed7) | SenseTalk (/wiki/Category:SenseTalk) | SETL (/wiki/Category:SETL)
| Sidef (/wiki/Category:Sidef) | Simula (/wiki/Category:Simula) | Sinclair ZX81 BASIC (/wiki/Category:Sinclair_ZX81_BASIC) | Slate (/wiki/Category:Slate)
| Smalltalk (/wiki/Category:Smalltalk) | Sparkling (/wiki/Category:Sparkling) | Spin (/wiki/Category:Spin) | SPL (/wiki/Category:SPL) | SQL PL (/wiki/Category:SQL_PL)
| Standard ML (/wiki/Category:Standard_ML) | Stata (/wiki/Category:Stata) | Suneido (/wiki/Category:Suneido) | Swift (/wiki/Category:Swift)
| Tailspin (/wiki/Category:Tailspin) | Tcl (/wiki/Category:Tcl) | PlainTeX (/wiki/Category:PlainTeX) | TI-83 BASIC (/wiki/Category:TI-83_BASIC)
| TI-89 BASIC (/wiki/Category:TI-89_BASIC) | TorqueScript (/wiki/Category:TorqueScript) | Transact-SQL (/wiki/Category:Transact-SQL) | Trith (/wiki/Category:Trith)
| TUSCRIPT (/wiki/Category:TUSCRIPT) | Uniface (/wiki/Category:Uniface) | UNIX Shell (/wiki/Category:UNIX_Shell) | UnixPipes (/wiki/Category:UnixPipes)
| Ursa (/wiki/Category:Ursa) | Ursala (/wiki/Category:Ursala) | V (/wiki/Category:V) | Vala (/wiki/Category:Vala) | VBA (/wiki/Category:VBA)
| Vedit macro language (/wiki/Category:Vedit_macro_language) | Verbexx (/wiki/Category:Verbexx) | Verilog (/wiki/Category:Verilog) | Vim Script (/wiki/Category:Vim_Script)
| Visual Basic .NET (/wiki/Category:Visual_Basic_.NET) | Wart (/wiki/Category:Wart) | Wee Basic (/wiki/Category:Wee_Basic) | Whitespace (/wiki/Category:Whitespace)
| Wren (/wiki/Category:Wren) | X86 Assembly (/wiki/Category:X86_Assembly) | XBasic (/wiki/Category:XBasic) | XLISP (/wiki/Category:XLISP) | XPL0 (/wiki/Category:XPL0)
| Yabasic (/wiki/Category:Yabasic) | Zkl (/wiki/Category:Zkl)