The Unshrinking and Unreducing C source code of Samuel H. Smith was used in UnZip 2.0. The full version 1.2 of Samuel H. Smith's code from the file ST_UNZIP.ZIP follows:

```
#define ATARI_ST 1

/*
 * Copyright 1989 Samuel H. Smith;   All rights reserved
 *
 * Do not distribute modified versions without my permission.
 * Do not remove or alter this notice or any other copyright notice.
 * If you use this in your own program you must distribute source code.
 * Do not use any of this in a commercial product.
 *
 */

/*
 * UnZip - A simple zipfile extract utility
 *
 * To compile:
 *         tcc -B -O -Z -G -mc unzip.c            ;turbo C 2.0, compact model
 *
 */

#define CODE_VERSION   "UnZip:   Zipfile Extract v1.2 of 03-15-89;   (C) 1989 Samuel H. Smith"
#define PRG_VERSION     "\
UnZip:   Atari ST Zipfile Extract v1.2 of 03-30-89\n\
Ported to the Atari ST by Darin Wayrynen.\n\
Derived from code Copyrighted (c)1989 by S.H.Smith"

typedef unsigned char byte;
/* code assumes UNSIGNED bytes */
typedef long longint;
typedef unsigned word;
typedef char boolean;

#define STRSIZ 256

#include <stdio.h>
 /* this is your standard header for all C compiles */

#ifndef ATARI_ST
#include <stdlib.h>
 /* this include defines various standard library prototypes */
#else
long buffer_size,in_size,out_size;
#endif

/*
 * SEE HOST OPERATING SYSTEM SPECIFICS SECTION STARTING NEAR LINE 180
 *
 */


/* --------------------------------------------------------- */
/*
 * Zipfile layout declarations
 *
 */

typedef longint signature_type;


#define LOCAL_FILE_HEADER_SIGNATURE   0x04034b50L


typedef struct local_file_header

word version_needed_to_extract;

word general_purpose_bit_flag;

word compression_method;

word last_mod_file_time;

word last_mod_file_date;

longint crc32;

longint compressed_size;

longint uncompressed_size;

word filename_length;

word extra_field_length;
 local_file_header;


#define CENTRAL_FILE_HEADER_SIGNATURE   0x02014b50L


typedef struct central_directory_file_header

word version_made_by;

word version_needed_to_extract;

word general_purpose_bit_flag;

word compression_method;

word last_mod_file_time;

word last_mod_file_date;

longint crc32;

longint compressed_size;
longint uncompressed_size;

word filename_length;

word extra_field_length;

word file_comment_length;

word disk_number_start;

word internal_file_attributes;

longint external_file_attributes;

longint relative_offset_local_header;
 central_directory_file_header;


#define END_CENTRAL_DIR_SIGNATURE   0x06054b50L


typedef struct end_central_dir_record

word number_this_disk;

word number_disk_with_start_central_directory;

word total_entries_central_dir_on_this_disk;

word total_entries_central_dir;

longint size_central_directory;

longint offset_start_central_directory;

word zipfile_comment_length;
 end_central_dir_record;



/* ---------------------------------------------------------- */
/*
 * input file variables
 *
 */

#define INBUFSIZ 0x2000L
byte *inbuf;


/* input file buffer - any size is legal */
byte *inptr;
```

```
int incnt;
unsigned bitbuf;
int bits_left;
boolean zipeof;

int zipfd;
char zipfn[STRSIZ];
local_file_header lrec;


/* ------------------------------------------------------------ */
/*
 * output stream variables
 *
 */

#define OUTBUFSIZ 0x6000L
byte *outbuf;                              /* buffer for rle look-back */
byte *outptr;

longint outpos;


/* absolute position in outfile */
int outcnt;


/* current position in outbuf */

int outfd;
char filename[STRSIZ];
char extra[STRSIZ];

#define DLE 144


/* ------------------------------------------------------------ */
/*
 * shrink/reduce working storage
 *
 */

int factor;
byte followers[256][64];
byte Slen[256];

#define max_bits 13
#define init_bits 9
#define hsize 8192
#define first_ent 257
#define clear 256

typedef int hsize_array_integer[hsize+1];
typedef byte hsize_array_byte[hsize+1];
```

```
hsize_array_integer prefix_of;
hsize_array_byte suffix_of;
hsize_array_byte stack;

int codesize;
int maxcode;
int free_ent;
int maxcodemax;
int offset;
int sizex;




/*
==================================================================
*/
/*
 * Host operating system details
 *
 */

#ifndef ATARI_ST
#include <string.h>
#else
#include <strings.h>
#define strchr index
#endif
 /* this include defines strcpy, strcmp, etc. */

#ifndef ATARI_ST
#include <io.h>
 /*
   * this include file defines
   *                   struct ftime ...            (* file time/date stamp info *)
   *                   int setftime (int handle, struct ftime *ftimep);
   *                   #define SEEK_CUR   1       (* lseek() modes *)
   *                   #define SEEK_END   2
   *                   #define SEEK_SET   0
   */
#else
#define SEEK_CUR   1       (* lseek() modes *)
#define SEEK_END   2
#define SEEK_SET   0
#endif



#include <fcntl.h>
 /*
   * this include file defines
   *                   #define O_BINARY 0x8000   (* no cr-lf translation *)
   * as used in the open() standard function
   */

#ifndef ATARI_ST
#include <sys/stat.h>
 /*
   * this include file defines
   *                   #define S_IREAD 0x0100   (* owner may read *)
```

```
  *                    #define S_IWRITE 0x0080 (* owner may write *)
  * as used in the creat() standard function
  */
#endif

#ifdef ATARI_ST
#define HIGH_LOW 1
#include <osbind.h>
typedef struct _dt                              /* My creation! DAW */

unsigned realdate;

unsigned realtime;
 dt;
#endif

 /*
  * change 'undef' to 'define' if your machine stores high order bytes in
  * lower addresses.
  */

#ifndef ATARI_ST
void set_file_time()
 /*
  * set the output file date/time stamp according to information from the
  * zipfile directory record for this file
  */


union
                        dt ft;            /* system file time record */



struct
                                    word ztime;        /* date and time words */
                                    word zdate;        /* .. same format as in .ZIP file */



 zt;

 td;



/*

 * set output file date and time - this is optional and can be

 * deleted if your compiler does not easily support setftime()

 */


td.zt.ztime = lrec.last_mod_file_time;

td.zt.zdate = lrec.last_mod_file_date;
```

```
setftime(outfd, &td.ft);

#else
#define set_file_time() ;
#endif


int create_output_file()
 /* return non-0 if creat failed */


/* create the output file with READ and WRITE permissions */
#ifndef ATARI_ST

outfd = creat(filename, S_IWRITE | S_IREAD);
#else

outfd = open(filename, O_CREAT | O_RDWR| O_BINARY);
#endif

if (outfd < 1)


printf("Can't create output: %s\n", filename);


return 1;





/*

 * close the newly created file and reopen it in BINARY mode to

 * disable all CR/LF translations

 */
#ifndef ATARI_ST

/*

 * Not neccessary with Atari ST.   Just open() in Binary mode.

 */

close(outfd);

outfd = open(filename, O_RDWR | O_BINARY);
#endif
```

```
/* write a single byte at EOF to pre-allocate the file */
            lseek(outfd, lrec.uncompressed_size - 1L, SEEK_SET);

write(outfd, "?", 1);

lseek(outfd, 0L, SEEK_SET);

return 0;



int open_input_file()
 /* return non-0 if creat failed */


/*

 * open the zipfile for reading and in BINARY mode to prevent cr/lf

 * translation, which would corrupt the bitstreams

 */


zipfd = open(zipfn, O_RDONLY | O_BINARY);

if (zipfd < 1)


printf("Can't open input file: %s\n", zipfn);


return (1);



return 0;



#ifdef HIGH_LOW

void swap_bytes(wordp)
word *wordp;
 /* convert intel style 'short int' variable to host format */


char *charp = (char *) wordp;

char temp;
```

```
temp = charp[0];

charp[0] = charp[1];

charp[1] = temp;


void swap_lbytes(longp)
longint *longp;
 /* convert intel style 'long' variable to host format */


char *charp = (char *) longp;

char temp[4];


temp[3] = charp[0];

temp[2] = charp[1];

temp[1] = charp[2];

temp[0] = charp[3];


charp[0] = temp[0];

charp[1] = temp[1];

charp[2] = temp[2];

charp[3] = temp[3];


#endif



/*
==================================================================
*/

int FillBuffer()
 /* fill input buffer if possible */


int readsize;

            if (lrec.compressed_size <= 0)
```

```
return incnt = 0;

            if (lrec.compressed_size > in_size)


readsize = in_size;

else
                        readsize = (int) lrec.compressed_size;

incnt = read(zipfd, inbuf, readsize);

            lrec.compressed_size -= incnt;

inptr = inbuf;

return incnt--;



int ReadByte(x)
unsigned *x;
 /* read a byte; return 8 if byte available, 0 if not */



if (incnt-- == 0)



if (FillBuffer() == 0)




return 0;



*x = *inptr++;

return 8;




/* ------------------------------------------------------------- */
static unsigned mask_bits[] =
            0,        0x0001, 0x0003, 0x0007, 0x000f,
                        0x001f, 0x003f, 0x007f, 0x00ff,
                        0x01ff, 0x03ff, 0x07ff, 0x0fff,
                        0x1fff, 0x3fff, 0x7fff, 0xffff
            ;



int FillBitBuffer(bits)
register int bits;
```

```
/* get the bits that are left and read the next word */

unsigned temp;
            register int result = bitbuf;

int sbits = bits_left;

bits -= bits_left;


/* read next word of input */

bits_left = ReadByte(&bitbuf);

bits_left += ReadByte(&temp);

bitbuf |= (temp << 8);

if (bits_left == 0)


zipeof = 1;


/* get the remaining bits */
            result = result | (int) ((bitbuf & mask_bits[bits]) << sbits);
            bitbuf >>= bits;
            bits_left -= bits;
            return result;


#define READBIT(nbits,zdest)  if (nbits <= bits_left)  zdest = (int)(bitbuf & mask_bits[nbits]); bitbuf
>>= nbits; bits_left -= nbits;  else zdest = FillBitBuffer(nbits);

/*
 * macro READBIT(nbits,zdest)
 *
 *         if (nbits <= bits_left)
 *               zdest = (int)(bitbuf & mask_bits[nbits]);
 *               bitbuf >>= nbits;
 *               bits_left -= nbits;
 *          else
 *               zdest = FillBitBuffer(nbits);
 *
 *
 */


/* ------------------------------------------------------------ */

#include "crc32.h"
```

```
/* ------------------------------------------------------------- */

void FlushOutput()
 /* flush contents of output buffer */


UpdateCRC(outbuf, outcnt);

write(outfd, outbuf, outcnt);

outpos += outcnt;

outcnt = 0;

outptr = outbuf;


#define OUTB(intc)  *outptr++=intc; if (++outcnt==out_size) FlushOutput();

/*
 *   macro OUTB(intc)
 *
 *         *outptr++=intc;
 *         if (++outcnt==out_size)
 *               FlushOutput();
 *
 *
 */


/* ----------------------------------------------------------- */

void LoadFollowers()

            register int x;
            register int i;


for (x = 255; x >= 0; x--)
                        READBIT(6,Slen[x]);


for (i = 0; i < Slen[x]; i++)
                                    READBIT(8,followers[x][i]);









/* ----------------------------------------------------------- */
```

```
/*
 * The Reducing algorithm is actually a combination of two
 * distinct algorithms.   The first algorithm compresses repeated
 * byte sequences, and the second algorithm takes the compressed
 * stream from the first algorithm and applies a probabilistic
 * compression method.
 */

int L_table[] = 0, 0x7f, 0x3f, 0x1f, 0x0f;

int D_shift[] = 0, 0x07, 0x06, 0x05, 0x04;
int D_mask[]   = 0, 0x01, 0x03, 0x07, 0x0f;

int B_table[] = 8, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5,


 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6,


 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,


 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7,


 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,


 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,


 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,


 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,


 8, 8, 8, 8;

/* ------------------------------------------------------------ */

void unReduce()
 /* expand probablisticly reduced data */

            register int lchar;
            register int nchar;
            int ExState;
            register int V;
            register int Len;

            factor = lrec.compression_method - 1;

ExState = 0;

lchar = 0;

LoadFollowers();

            while (((outpos + outcnt) < lrec.uncompressed_size) && (!zipeof))


if (Slen[lchar] == 0)
                                    READBIT(8,nchar)         /* ; */
                        else



                                    READBIT(1,nchar);
                                    if (nchar != 0)
                                                READBIT(8,nchar)         /* ; */
                                    else




                                                int follower;
                                                int bitsneeded = B_table[Slen[lchar]];
                                                READBIT(bitsneeded,follower);
                                                nchar = followers[lchar][follower];
```

```
/* expand the resulting byte */

switch (ExState)


case 0:
                              if (nchar != DLE)
                                        OUTB(nchar) /*;*/



else



ExState = 1;


break;


case 1:
                              if (nchar != 0)
                                        V = nchar;



Len = V & L_table[factor];



if (Len == L_table[factor])




ExState = 2;
```

```
else

ExState = 3;

else
                    OUTB(DLE);

ExState = 0;

break;
          case 2:
                    Len += nchar;

ExState = 3;

break;
          case 3:
```

```
register int i = Len + 3;

int offset = (((V >> D_shift[factor]) &
                                        D_mask[factor]) << 8) + nchar + 1;

longint op = outpos + outcnt - offset;

/* special case- before start of file */

while ((op < 0L) && (i > 0))

OUTB(0);

op++;

i--;
```

```
/* normal copy of data from output buffer */
```

```
register int ix = (int) (op % out_size);
```

```
/* do a block memory copy if possible */
if ( ((ix      +i) < out_size) &&
        ((outcnt+i) < out_size) )
             memcpy(outptr,&outbuf[ix],i);
             outptr += i;
             outcnt += i;


/* otherwise copy byte by byte */
else while (i--)
             OUTB(outbuf[ix]);
             if (++ix >= out_size)
                          ix = 0;
```

```
ExState = 0;
```

```
break;
```

```
                        /* store character for next iteration */
                        lchar = nchar;




/* ---------------------------------------------------------- */
/*
 * Shrinking is a Dynamic Ziv-Lempel-Welch compression algorithm
 * with partial clearing.
 *
 */

void partial_clear()

            register int pr;
            register int cd;


/* mark all nodes as potentially unused */

for (cd = first_ent; cd < free_ent; cd++)


prefix_of[cd] |= 0x8000;


/* unmark those that are used by other nodes */

for (cd = first_ent; cd < free_ent; cd++)


pr = prefix_of[cd] & 0x7fff;
/* reference to another node? */
                        if (pr >= first_ent)                  /* flag node as referenced */



prefix_of[pr] &= 0x7fff;





/* clear the ones that are still marked */

for (cd = first_ent; cd < free_ent; cd++)


if ((prefix_of[cd] & 0x8000) != 0)
```

```
prefix_of[cd] = -1;


/* find first cleared node as next free_ent */
            cd = first_ent;
            while ((cd < maxcodemax) && (prefix_of[cd] != -1))
                        cd++;
            free_ent = cd;



/* ------------------------------------------------------------- */

void unShrink()

            #define   GetCode(dest) READBIT(codesize,dest)


register int code;

register int stackp;

int finchar;

int oldcode;

int incode;



/* decompress the file */

maxcodemax = 1 << max_bits;

codesize = init_bits;

maxcode = (1 << codesize) - 1;

free_ent = first_ent;

offset = 0;

sizex = 0;


for (code = maxcodemax; code > 255; code--)


prefix_of[code] = -1;


for (code = 255; code >= 0; code--)
```

```
prefix_of[code] = 0;

suffix_of[code] = code;



GetCode(oldcode);

if (zipeof)

return;

finchar = oldcode;

        OUTB(finchar);

        stackp = hsize;

while (!zipeof)

GetCode(code);

if (zipeof)

return;

while (code == clear)

GetCode(code);

switch (code)
```

```
case 1:

codesize++;

if (codesize == max_bits)

maxcode = maxcodemax;

else

maxcode = (1 << codesize) - 1;

break;
```

```
case 2:

partial_clear();

break;

GetCode(code);

if (zipeof)

return;

/* special case for KwKwK string */

incode = code;

if (prefix_of[code] == -1)
                    stack[--stackp] = finchar;

code = oldcode;
```

```
/* generate output characters in reverse order */


while (code >= first_ent)
                              stack[--stackp] = suffix_of[code];



code = prefix_of[code];





finchar = suffix_of[code];
                    stack[--stackp] = finchar;


                    /* and put them out in forward order, block copy */
                    if ((hsize-stackp+outcnt) < out_size)
                              memcpy(outptr,&stack[stackp],hsize-stackp);
                              outptr += hsize-stackp;
                              outcnt += hsize-stackp;
                              stackp = hsize;


                    /* output byte by byte if we can't go by blocks */
                    else while (stackp < hsize)
                              OUTB(stack[stackp++]);




/* generate new entry */


code = free_ent;


if (code < maxcodemax)



prefix_of[code] = oldcode;
```

```
suffix_of[code] = finchar;

do

code++;

while ((code < maxcodemax) && (prefix_of[code] != -1));

free_ent = code;

/* remember previous code */

oldcode = incode;

/* ------------------------------------------------------------ */
void extract_member()

unsigned b;

bits_left = 0;
```

```
bitbuf = 0;

incnt = 0;

outpos = 0L;

outcnt = 0;

outptr = outbuf;

zipeof = 0;

crc32val = 0xFFFFFFFFL;


/* create the output file with READ and WRITE permissions */

if (create_output_file())


prg_exit(1);

        switch (lrec.compression_method)


case 0:

/* stored */




printf(" Extracting: %-12s ", filename);


while (ReadByte(&b))



OUTB(b);




break;
```

```
        case 1:

printf("UnShrinking: %-12s ", filename);

unShrink();



break;

case 2:

case 3:

case 4:
        case 5:

printf("   Expanding: %-12s ", filename);

unReduce();



break;

default:

printf("Unknown compression method.");



/* write the last partial buffer, if any */
```

```
if (outcnt > 0)


UpdateCRC(outbuf, outcnt);


write(outfd, outbuf, outcnt);




/* set output file date and time */

set_file_time();


close(outfd);


crc32val = -1 - crc32val;
#ifdef HIGH_LOW

swap_lbytes(&lrec.crc32);
#endif
        if (crc32val != lrec.crc32)
                printf(" Bad CRC %08lx   (should be %08lx)", lrec.crc32, crc32val);


printf("\n");




/* ------------------------------------------------------------ */

void get_string(len,s)
int len;
char *s;


read(zipfd, s, len);

s[len] = 0;




/* ------------------------------------------------------------ */

void process_local_file_header()


read(zipfd, &lrec, sizeof(lrec));
```

```
#ifdef HIGH_LOW

swap_bytes(&lrec.filename_length);

swap_bytes(&lrec.extra_field_length);

swap_lbytes(&lrec.compressed_size);

swap_lbytes(&lrec.uncompressed_size);

swap_bytes(&lrec.compression_method);
#endif


get_string(lrec.filename_length, filename);

get_string(lrec.extra_field_length, extra);

extract_member();



/* ---------------------------------------------------------- */

void process_central_file_header()


central_directory_file_header rec;

char filename[STRSIZ];

char extra[STRSIZ];

char comment[STRSIZ];


read(zipfd, &rec, sizeof(rec));

#ifdef HIGH_LOW

swap_bytes(&rec.filename_length);

swap_bytes(&rec.extra_field_length);

swap_bytes(&rec.file_comment_length);
#endif

      get_string(rec.filename_length, filename);

get_string(rec.extra_field_length, extra);
get_string(rec.file_comment_length, comment);



/* ---------------------------------------------------------- */

void process_end_central_dir()


end_central_dir_record rec;

char comment[STRSIZ];


read(zipfd, &rec, sizeof(rec));

#ifdef HIGH_LOW

swap_bytes(&rec.zipfile_comment_length);
#endif


get_string(rec.zipfile_comment_length, comment);



/* ---------------------------------------------------------- */

void process_headers()


longint sig;


while (1)


if (read(zipfd, &sig, sizeof(sig)) != sizeof(sig))



return;

#ifdef HIGH_LOW


swap_lbytes(&sig);
#endif

                        if (sig == LOCAL_FILE_HEADER_SIGNATURE)
```

```
 */

process_local_file_header();
                        else if (sig == CENTRAL_FILE_HEADER_SIGNATURE)

process_central_file_header();
                        else if (sig == END_CENTRAL_DIR_SIGNATURE)

process_end_central_dir();

return;

                        else

printf("Invalid Zipfile Header\n");

return;

/* ---------------------------------------------------------- */

void extract_zipfile()

/*

 * open the zipfile for reading and in BINARY mode to prevent cr/lf

 * translation, which would corrupt the bitstreams

 */
```

```
printf("Buffers - ");

printf("(input --> %7ld bytes) ",in_size);

printf("(output --> %7ld bytes)\n\n",out_size);


if (open_input_file())


prg_exit(1);


process_headers();


close(zipfd);




/* ---------------------------------------------------------- */
/*
 * main program
 *
 */

void main(argc,argv)
int argc;
char **argv;

#ifdef ATARI_ST

int buffer_fail = 0;
#endif




printf("\n\n%s\n",PRG_VERSION);


printf("Courtesy of:   Darin Wayrynen, S.H.Smith and\n                    The Tool Shop BBS,   (602) 279-
2673.\n");

if (argc != 2)





printf("\nYou may copy and distribute this program freely, provided that:\n");
printf("      1)     No fee is charged for such copying and distribution, and\n");


printf("      2)     It is distributed ONLY in its original, unmodified state.\n\n");


printf("If you wish to distribute a modified version of this program, you MUST\n");


printf("include the source code.\n\n");


printf("If you modify this program, we would appreciate a copy of the   new source\n");


printf("code.   Samuel is holding the copyright on the source code, so please don't\n");


printf("delete his   name from the program files or from the documentation.\n\n");
            printf("IN NO EVENT WILL WE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST\n");
            printf("PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES\n");
            printf("ARISING OUT OF YOUR USE OR INABILITY TO USE THE PROGRAM, OR FOR ANY\n");
            printf("CLAIM BY ANY OTHER PARTY.\n\n");
            printf("Usage:   UnZip FILE[.zip]\n");
#ifdef ATARI_ST


prg_exit(0);
#else
            exit(1);
#endif




printf("\n");



/* .ZIP default if none provided by user */

strcpy(zipfn, argv[1]);

if (strchr(zipfn, '.') == NULL)


strcat(zipfn, ".ZIP");

#ifdef ATARI_ST

/* ST buffer allocation */
```

```
buffer_size = Malloc(-1L)-10000L;                    /* returns largest free block */

if (buffer_size<(INBUFSIZ + OUTBUFSIZ))


buffer_fail=1;

in_size = (buffer_size-(INBUFSIZ + OUTBUFSIZ))/100L*55L+INBUFSIZ;

out_size = (buffer_size-(INBUFSIZ + OUTBUFSIZ))/100L*35L+OUTBUFSIZ;

inbuf = (byte *)Malloc(in_size);

/* 60% */

outbuf = (byte *)Malloc(out_size);

/* 40% */

if ((inbuf== NULL) || (outbuf == NULL) || buffer_fail)




printf("Can't allocate buffers!\n");


prg_exit(1);


#else
        /* allocate i/o buffers */

inbuf = (byte *) (malloc(INBUFSIZ));

outbuf = (byte *) (malloc(OUTBUFSIZ));

if ((inbuf == NULL) || (outbuf == NULL))


printf("Can't allocate buffers!\n");


exit(1);


#endif
        /* do the job... */
        extract_zipfile();
#ifdef ATARI_ST

prg_exit(0);
```

```
#else

exit(0);
#endif


prg_exit(value)
int value;


setbuffer(stdout,0L);





/* turn off printf













   buffering */

printf("\nPress any key to continue.");



Bconin(2);







/* wait for keypress before

                                                  exiting, in case prg is

                                                  run from desktop */
```

```
printf("\n");

exit(value);



#ifdef ATARI_ST
memcpy(to,from,count)
register char *to,*from;
register unsigned int count;



for(;count>0;--count,++to,++from)



*to=*from;

#endif
```