Smally

P. Baillehache

June 10, 2020

Contents

1 Interface **1**

2 Code **4**
2.1 smally.c 4
2.2 smally-inline.c 13

3 Makefile **16**

4 Unit tests **17**

5 Unit tests output **18**

6 Tool **18**

Introduction

Smally library is a C library to compress and decompress data.

It provides an implementation of the LZ77 algorithm.

It uses the `PBErr` and `GSet` libraries.

1 Interface

```
// *************** CRYPTIC.H ***************
#ifndef CRYPTIC_H
#define CRYPTIC_H
```

```
// ================= Include ==================
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbmath.h"
#include "gset.h"

// ================= Define ==================

#define SMALLYLZ77_DEFAULT_SIZELEN 12   // bits
#define SMALLYLZ77_DEFAULT_SIZEPOS 4    // bits

// ================= Data structures ===================

// Operating mode
typedef enum SmallyType {

  SmallyType_LZ77

} SmallyType;

// Structure for the Smally
typedef struct Smally {

  // Type
  SmallyType type;

} Smally;

// https://towardsdatascience.com/how-data-compression-works-
// exploring-lz77-3a2c2e06c097

// Structure for the SmallyLZ77
typedef struct SmallyLZ77 {

  // Parent
  Smally parent;

  // Size in bits of the search buffer
  unsigned int nbBitSearchBuffer;

  // Size in bits of the lookahead buffer
  unsigned int nbBitLookAheadBuffer;

  // Size in of the search buffer
  unsigned int sizeSearchBuffer;

  // Size in of the lookahead buffer
  unsigned int sizeLookAheadBuffer;

} SmallyLZ77;

// ================ Functions declaration ====================

// Static constructor for a Smally of type 'type'
#if BUILDMODE != 0
static inline
#endif
Smally SmallyCreateStatic(SmallyType type);

// Function to free the memory used by the static Smally
```

```
void _SmallyFreeStatic(Smally* that);

// Get the type of the Smally 'that'
#if BUILDMODE != 0
static inline
#endif
SmallyType _SmallyGetType(const Smally* const that);

// Static constructor for a SmallyLZ77
#if BUILDMODE != 0
static inline
#endif
SmallyLZ77 SmallyLZ77CreateStatic(void);

// Function to free the memory used by the static SmallyLZ77
void SmallyLZ77FreeStatic(SmallyLZ77* that);

// Get the nb of bits for the search buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetNbBitSearchBuffer(const SmallyLZ77* const that);

// Get the nb of bits for the look ahead buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetNbBitLookAheadBuffer(const SmallyLZ77* const that);

// Get the size of the search buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetSizeSearchBuffer(const SmallyLZ77* const that);

// Get the size of the look ahead buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetSizeLookAheadBuffer(const SmallyLZ77* const that);

// Function to compress a file 'fpIn' with the SmallyLZ77 'that'
// Save the result in the file 'fpOut'.
void _SmallyLZ77CompressFile(
  const SmallyLZ77* const that,
              FILE* const fpIn,
              FILE* const fpOut);

// Function to decompress a file 'fpIn' with the SmallyLZ77 'that'
// Save the result in the file 'fpOut'.
void _SmallyLZ77DecompressFile(
  const SmallyLZ77* const that,
              FILE* const fpIn,
              FILE* const fpOut);

// ================ inliner ====================

#if BUILDMODE != 0
#include "smally-inline.c"
#endif

#endif
```

```
// ================= Generic functions ==================

#define SmallyGetType(S) _Generic(S, \
  Smally*: _SmallyGetType, \
  SmallyLZ77*: _SmallyGetType, \
  default: PBErrInvalidPolymorphism)((const Smally*)S)

#define SmallyFreeStatic(S) _Generic(S, \
  Smally*: _SmallyFreeStatic, \
  SmallyLZ77*: SmallyLZ77FreeStatic, \
  default: PBErrInvalidPolymorphism)(S)

#define SmallyCompressFile(S, I, O) _Generic(S, \
  SmallyLZ77*: _SmallyLZ77CompressFile, \
  const SmallyLZ77*: _SmallyLZ77CompressFile, \
  default: PBErrInvalidPolymorphism)(S, I, O)

#define SmallyDecompressFile(S, I, O) _Generic(S, \
  SmallyLZ77*: _SmallyLZ77DecompressFile, \
  const SmallyLZ77*: _SmallyLZ77DecompressFile, \
  default: PBErrInvalidPolymorphism)(S, I, O)
```

2 Code

2.1 smally.c

```
// *************** CRYPTIC.C ***************

// ================= Include ==================
#include "smally.h"
#if BUILDMODE == 0
#include "smally-inline.c"
#endif

// ================= Data structures ===================

typedef struct SmallyLZ77Token {

  unsigned int offset;
  unsigned int length;
  unsigned char breakChar;

} SmallyLZ77Token;

// ================ Functions declaration ====================

// Function to search a token for the LZ77 algorithm
SmallyLZ77Token SmallyLZ77SearchToken(
  const SmallyLZ77* const that,
        const GSet* const searchSet,
        const GSet* const lookAheadSet);

// ================ Functions implementation ==================

// Function to free the memory used by the static Smally
void _SmallyFreeStatic(Smally* that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  (void)that;

}

// Function to free the memory used by the static SmallyLZ77
void SmallyLZ77FreeStatic(SmallyLZ77* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  // Free the parent
  SmallyFreeStatic((Smally*)that);

}

// Function to compress a file 'fpIn' with the Smally 'that'
// Save the result in the file 'fpOut'.
void _SmallyLZ77CompressFile(
  const SmallyLZ77* const that,
              FILE* const fpIn,
              FILE* const fpOut) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

  if (fpIn == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
```

```
    sprintf(
      SmallyErr->_msg,
      "'fpIn' is null");
    PBErrCatch(SmallyErr);

  }

  if (fpOut == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'fpOut' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  // Create a dictionary of all possible values to be used as data
  // of the GSet
  unsigned char vals[256];
  for (
    int i = 256;
    i--;) {

    vals[i] = i;

  }

  // Declare the search and look ahead buffers
  GSet searchSet = GSetCreateStatic();
  GSet lookAheadSet = GSetCreateStatic();

  // Loop until the look ahead buffer is not empty or the file is not
  // completely processed
  do {

    // Refill the look ahead buffer
    while (
      !feof(fpIn) &&
      GSetNbElem(&lookAheadSet) < SmallyGetSizeLookAheadBuffer(that) - 1) {

      unsigned char c;
      size_t ret =
        fread(
          &c,
          1,
          1,
          fpIn);
      if (ret == 1) {

        GSetAppend(
          &lookAheadSet,
          vals + c);

      } else if (!feof(fpIn)) {

        SmallyErr->_type = PBErrTypeIOError;
        sprintf(
          SmallyErr->_msg,
          "I/O error %d",
```

```
      ferror(fpIn));
    PBErrCatch(SmallyErr);

  }

}

// Search the longest token in the dictionnary
SmallyLZ77Token token;
if (GSetNbElem(&searchSet) > 0) {

  token =
    SmallyLZ77SearchToken(
      that,
      &searchSet,
      &lookAheadSet);

} else {

  token = (SmallyLZ77Token) {

    .length = 0,
    .offset = 0,
    .breakChar = *(unsigned char*)GSetHead(&lookAheadSet)

  };

}

// Write the token (l, o, c) to the output file

unsigned char buffer[3];
buffer[2] = token.breakChar;
buffer[1] = token.offset & 0xFF;
buffer[0] = (token.offset & 0xFF00) >> 8;
buffer[0] |= token.length << (8 - SmallyGetNbBitLookAheadBuffer(that));

size_t ret =
  fwrite(
    buffer,
    1,
    3,
    fpOut);
if (
  ret != 3 &&
  !feof(fpOut)) {

  SmallyErr->_type = PBErrTypeIOError;
  sprintf(
    SmallyErr->_msg,
    "I/O error %d",
    ferror(fpOut));
  PBErrCatch(SmallyErr);

}

// Slide the window
// First, move the bytes from the look ahead buffer ot the search
// buffer
for (
  unsigned int i = token.length + 1;
  i--;) {
```

```
      GSetAppend(
        &searchSet,
        GSetPop(&lookAheadSet));

    }

    // Second, pop from the search buffer as long as it exceeds its maximum
    // size
    while (GSetNbElem(&searchSet) >= SmallyGetSizeSearchBuffer(that)) {

      (void)GSetPop(&searchSet);

    }

  } while (
    !feof(fpIn) ||
    GSetNbElem(&lookAheadSet) > 0);

  // Free memory
  GSetFlush(&searchSet);
  GSetFlush(&lookAheadSet);

}

// Function to search a token for the LZ77 algorithm
SmallyLZ77Token SmallyLZ77SearchToken(
  const SmallyLZ77* const that,
        const GSet* const searchSet,
        const GSet* const lookAheadSet) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

  if (searchSet == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'searchSet' is null");
    PBErrCatch(SmallyErr);

  }

  if (lookAheadSet == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'lookAheadSet' is null");
    PBErrCatch(SmallyErr);

  }
```

```
#endif

  (void)that;

  // Create the result token
  SmallyLZ77Token bestToken = {

    .length = 0,
    .offset = 0,
    .breakChar = *(unsigned char*)GSetHead(lookAheadSet)

  };

  // Create an iterator for the look ahead buffer
  GSetIterForward iterLookAhead =
    GSetIterForwardCreateStatic(lookAheadSet);

  // Create an iterator for the start in search buffer
  GSetIterBackward iterStartSearch =
    GSetIterBackwardCreateStatic(searchSet);

  // Create an iterator to scan the search buffer
  GSetIterForward iterSearch =
    GSetIterForwardCreateStatic(searchSet);

  // Declare a variable to memorize the offset
  unsigned int offset = 1;

  // Loop until we have scanned the whole dictionary
  do {

    // Create a temporary token
    SmallyLZ77Token token = {

      .length = 0,
      .offset = offset - 1,
      .breakChar = *(unsigned char*)GSetHead(lookAheadSet)

    };

    // Reset the iterator of the look ahead buffer
    GSetIterReset(&iterLookAhead);

    // Reset the iterator of the scan in the dictionnary
    memcpy(
      &iterSearch,
      &iterStartSearch,
      sizeof(GSetIterForward));

    // Scan the look ahead and dictionnary buffers until
    // we found a different byte
    bool flagEqual = false;

    do {

      // If the current byte in the look ahead buffer and search buffer
      // are equals
      flagEqual =
        (GSetIterGet(&iterSearch) == GSetIterGet(&iterLookAhead));
      if (
        flagEqual &&
```

```
        !GSetIterIsLast(&iterLookAhead)) {

        // Increase the length of the token
        ++(token.length);

      // Else, if we have found the breaking byte
      } else {

        token.breakChar = *(unsigned char*)GSetIterGet(&iterLookAhead);

      }

    } while (
      flagEqual &&
      GSetIterStep(&iterLookAhead) &&
      GSetIterStep(&iterSearch));

    // If the bytes were equals up to the end of the search of look
    // ahead buffer
    if (flagEqual) {

      // The breaking byte is set to the current byte in the
      // look ahead buffer
      token.breakChar = *(unsigned char*)GSetIterGet(&iterLookAhead);

    }

    // If the token is longer than the current best one
    if (bestToken.length < token.length) {

      // Update the best token
      bestToken = token;

    }

    // Increment the offset
    ++offset;

  } while (GSetIterStep(&iterStartSearch));

  // Return the result token
  return bestToken;

}

// Function to decompress a file 'fpIn' with the SmallyLZ77 'that'
// Save the result in the file 'fpOut'.
void _SmallyLZ77DecompressFile(
  const SmallyLZ77* const that,
              FILE* const fpIn,
              FILE* const fpOut) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);
```

```
  }

  if (fpIn == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'fpIn' is null");
    PBErrCatch(SmallyErr);

  }

  if (fpOut == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'fpOut' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  // Create a dictionary of all possible values to be used as data
  // of the GSet
  unsigned char vals[256];
  for (
    int i = 256;
    i--;) {

    vals[i] = i;

  }

  // Declare the search and look ahead buffers
  GSet searchSet = GSetCreateStatic();

  // Read the first token (l, o, c) from the input file
  unsigned char buffer[3];
  size_t ret =
    fread(
      buffer,
      1,
      3,
      fpIn);
  if (
    ret != 3 &&
    !feof(fpIn)) {

    SmallyErr->_type = PBErrTypeIOError;
    sprintf(
      SmallyErr->_msg,
      "I/O error %d",
      ferror(fpIn));
    PBErrCatch(SmallyErr);

  }

  // Loop on the file
  do {
```

```
SmallyLZ77Token token;
token.breakChar = buffer[2];
token.offset = (unsigned int)buffer[1];
int shift = (8 - SmallyGetNbBitLookAheadBuffer(that));
token.length = (unsigned int)(buffer[0] >> shift);
buffer[0] = buffer[0] << SmallyGetNbBitLookAheadBuffer(that);
token.offset += 256 * (unsigned int)
  (buffer[0] >> SmallyGetNbBitLookAheadBuffer(that));

// Decode the token
GSetIterBackward iterOffset = GSetIterBackwardCreateStatic(&searchSet);
for (
  unsigned int i = token.offset;
  i--;) {

    GSetIterStep(&iterOffset);

}

GSetIterForward iterLength;
memcpy(
  &iterLength,
  &iterOffset,
  sizeof(GSetIterForward));
for (
  unsigned int i = token.length;
  i--;) {

    unsigned char c = *(unsigned char*)GSetIterGet(&iterLength);
    ret =
      fwrite(
        &c,
        1,
        1,
        fpOut);
    if (
      ret != 1 &&
      !feof(fpOut)) {

      SmallyErr->_type = PBErrTypeIOError;
      sprintf(
        SmallyErr->_msg,
        "I/O error %d",
        ferror(fpOut));
      PBErrCatch(SmallyErr);

    }

    GSetAppend(
      &searchSet,
      vals + c);
    GSetIterStep(&iterLength);

}

// Output the breaking byte
ret =
  fwrite(
    &(token.breakChar),
    1,
    1,
    fpOut);
```

```
    if (
      ret != 1 &&
      !feof(fpOut)) {

      SmallyErr->_type = PBErrTypeIOError;
      sprintf(
        SmallyErr->_msg,
        "I/O error %d",
        ferror(fpOut));
      PBErrCatch(SmallyErr);

    }

    GSetAppend(
      &searchSet,
      vals + token.breakChar);

    // Pop from the search buffer as long as it exceeds its maximum
    // size
    while (GSetNbElem(&searchSet) >= SmallyGetSizeSearchBuffer(that)) {

      (void)GSetPop(&searchSet);

    }

    // Read the next token
    ret =
      fread(
        buffer,
        1,
        3,
        fpIn);
    if (
      ret != 3 &&
      !feof(fpIn)) {

      SmallyErr->_type = PBErrTypeIOError;
      sprintf(
        SmallyErr->_msg,
        "I/O error %d",
        ferror(fpIn));
      PBErrCatch(SmallyErr);

    }

  } while (!feof(fpIn));

  // Free memory
  GSetFlush(&searchSet);

}
```

2.2 smally-inline.c

```
// *************** CRYPTIC-INLINE.C ***************

// ================ Functions implementation ====================

// Static constructor for a Smally of type 'type'
#if BUILDMODE != 0
```

```
static inline
#endif
Smally SmallyCreateStatic(SmallyType type) {

  // Declare a Smally and set the properties
  Smally s = {

    .type = type

  };

  // Return the Smally
  return s;

}

// Get the type of the Smally 'that'
#if BUILDMODE != 0
static inline
#endif
SmallyType _SmallyGetType(const Smally* const that) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  // Return the type
  return that->type;

}

// Static constructor for a SmallyLZ77
#if BUILDMODE != 0
static inline
#endif
SmallyLZ77 SmallyLZ77CreateStatic(void) {

  // Declare a Smally and set the properties
  SmallyLZ77 s = {

    .nbBitSearchBuffer = SMALLYLZ77_DEFAULT_SIZELEN,
    .nbBitLookAheadBuffer = SMALLYLZ77_DEFAULT_SIZEPOS

  };

  s.parent = SmallyCreateStatic(SmallyType_LZ77);
  s.sizeSearchBuffer =
    powi(
      2,
      s.nbBitSearchBuffer);
  s.sizeLookAheadBuffer =
    powi(
```

```
      2,
      s.nbBitLookAheadBuffer);

  // Return the Smally
  return s;

}

// Get the nb of bits for the search buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetNbBitSearchBuffer(const SmallyLZ77* const that) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  return that->nbBitSearchBuffer;

}

// Get the nb of bits for the look ahead buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetNbBitLookAheadBuffer(const SmallyLZ77* const that) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  return that->nbBitLookAheadBuffer;

}

// Get the size of the search buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetSizeSearchBuffer(const SmallyLZ77* const that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  return that->sizeSearchBuffer;

}

// Get the size of the look ahead buffer of the SmallyLZ77 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int SmallyGetSizeLookAheadBuffer(const SmallyLZ77* const that) {

#if BUILDMODE == 0

  if (that == NULL) {

    SmallyErr->_type = PBErrTypeNullPointer;
    sprintf(
      SmallyErr->_msg,
      "'that' is null");
    PBErrCatch(SmallyErr);

  }

#endif

  return that->sizeLookAheadBuffer;

}
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main smally

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Check code style
style:
cbo *.h *.c
```

```
# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=smally
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$

# Rules to make the tool
smally: \
main-smally.o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) main-smally.o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG) -o smal

main-smally.o: \
main-smally.c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c main-smally.c

install:
cp smally ~/Tools/smally

testSmally:
smally -lz77 -out test.sma -compress main.c && smally -lz77 -out test.txt -decompress test.sma && diff main.c test.tx
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "smally.h"

void UnitTestSmallyLZ77() {

  SmallyLZ77 smally = SmallyLZ77CreateStatic();
  FILE* fpIn =
    fopen(
      "./testSmallyLZ77.txt",
      "r");
  FILE* fpOut =
    fopen(
      "./testSmallyLZ77.sma",
      "w");
  SmallyCompressFile(
    &smally,
```

```
    fpIn,
    fpOut);
  fclose(fpIn);
  fclose(fpOut);
  fpOut =
    fopen(
      "./testSmallyLZ77.sma",
      "r");
  FILE* fpBack =
    fopen(
      "./testSmallyLZ77.decomp.txt",
      "w");
  SmallyDecompressFile(
    &smally,
    fpOut,
    fpBack);
  fclose(fpOut);
  fclose(fpBack);
  SmallyFreeStatic(&smally);
  printf("UnitTestSmallyLZ77 OK\n");

}

void UnitTestAll() {

  UnitTestSmallyLZ77();
  printf("UnitTestAll OK\n");

}

int main() {

  UnitTestAll();

  // Return success code
  return 0;

}
```

5 Unit tests output

```
UnitTestSmallyLZ77 OK
UnitTestAll OK
```

6 Tool

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "smally.h"

int main(
     int argc,
  char** argv) {

  // Declare variables to process arguments
  FILE* fpOut = NULL;
```

```
FILE* fpIn = NULL;
SmallyType opMode = SmallyType_LZ77;

// Loop on arguments
for (
  int iArg = 1;
  iArg < argc;
  ++iArg) {

  int retStrCmp =
    strcmp(
      argv[iArg],
      "-help");
  if (retStrCmp == 0) {

    printf("smally\n");
    printf("[-help] : print the help message\n");
    printf("[-out <path>] : Save the result to the file at <path>\n");
    printf("                If not specified uses stdout\n");
    printf("[-compress <path>] : Encode the file at <path>\n");
    printf("[-decompress <path>] : Decode the file at <path>\n");
    printf("[-lz77] : Use LZ77 compression\n");
    printf("\n");

  // Else
  } else {

    // If the argument is -out
    unsigned int retStrCmp =
      strcmp(
        argv[iArg],
        "-out");
    if (retStrCmp == 0) {

      // If the output file is opened
      if (fpOut != NULL) {

        // Close it
        fclose(fpOut);

      }

      // Open the file
      fpOut =
        fopen(
          argv[iArg + 1],
          "w");

    }

    // If the argument is -compress
    retStrCmp =
      strcmp(
        argv[iArg],
        "-compress");
    if (retStrCmp == 0) {

      // If the input file is opened
      if (fpIn != NULL) {

        // Close it
        fclose(fpIn);
```

```
  }

  // Open the file
  fpIn =
    fopen(
      argv[iArg + 1],
      "r");

  // If the path is incorrect
  if (fpIn == NULL) {

    printf(
      "The path [%s] is incorrect\n",
      argv[iArg + 1]);
    return 1;

  }

  // If the output file is not specified
  if (fpOut == NULL) {

    fpOut =
      fopen(
        "/dev/stdout",
        "w");

  }

  if (opMode == SmallyType_LZ77) {

    // Create the Smally
    SmallyLZ77 smally = SmallyLZ77CreateStatic();

    // Compress the file
    SmallyCompressFile(
      &smally,
      fpIn,
      fpOut);

    // Free memory
    SmallyFreeStatic(&smally);

  }

}

// If the argument is -decompress
retStrCmp =
  strcmp(
    argv[iArg],
    "-decompress");
if (retStrCmp == 0) {

  // If the input file is opened
  if (fpIn != NULL) {

    // Close it
    fclose(fpIn);

  }
```

```
      // Open the file
      fpIn =
        fopen(
          argv[iArg + 1],
          "r");

      // If the path is incorrect
      if (fpIn == NULL) {

        printf(
          "The path [%s] is incorrect\n",
          argv[iArg + 1]);
        return 1;

      }

      // If the output file is not specified
      if (fpOut == NULL) {

        fpOut =
          fopen(
            "/dev/stdout",
            "w");

      }

      if (opMode == SmallyType_LZ77) {

        // Create the Smally
        SmallyLZ77 smally = SmallyLZ77CreateStatic();

        // Decompress the file
        SmallyDecompressFile(
          &smally,
          fpIn,
          fpOut);

        // Free memory
        SmallyFreeStatic(&smally);

      }

    }

    // If the argument is -lz77
    retStrCmp =
      strcmp(
        argv[iArg],
        "-lz77");
    if (retStrCmp == 0) {

      // Memorize the operation mode
      opMode = SmallyType_LZ77;

    }

  }

}

// If the input file is opened
if (fpIn != NULL) {
```

```
    // Close it
    fclose(fpIn);

  }

  // If the output file is opened
  if (fpOut != NULL) {

    // Close it
    fclose(fpOut);

  }

  // Return success code
  return 0;

}
```