```python
#!/usr/bin/python3

"""
WARNING: The code you are about to view is DISGUSTING

I wrote most of it months ago, so don't ask me what it's doing, or why.
"""

import struct
import sys
from ctypes import *


# BEGIN ZIP FILE STRUCTURES

class LocalFileHeader(Structure):
	_pack_ = 1
	_fields_ = [
		("magic", c_uint32),
		("version_needed", c_uint16),
		("flags", c_uint16),
		("compression_method", c_uint16),
		("mtime", c_uint16),
		("mdate", c_uint16),
		("crc32", c_uint32),
		("comp_size", c_uint32),
		("uncomp_size", c_uint32),
		("filename_len", c_uint16),
		("extra_len", c_uint16)]

class CentralDirectory(Structure):
	_pack_ = 1
	_fields_ = [
		("magic", c_uint32),
		("version_made", c_uint16),
		("version_needed", c_uint16),
		("flags", c_uint16),
		("compression_method", c_uint16),
		("mtime", c_uint16),
		("mdate", c_uint16),
		("crc32", c_uint32),
		("comp_size", c_uint32),
		("uncomp_size", c_uint32),
		("filename_len", c_uint16),
		("extra_len", c_uint16),
		("comment_len", c_uint16),
		("disk_num", c_uint16),
		("int_file_attr", c_uint16),
		("ext_file_attr", c_uint32),
		("local_header_offset", c_uint32)]

class EndCentralDirectory(Structure):
	_pack_ = 1
	_fields_ = [
		("magic", c_uint32),
		("disk_num", c_uint16),
		("start_disk", c_uint16),
		("num_entries_disk", c_uint16),
		("num_entries_total", c_uint16),
		("central_dir_size", c_uint32),
		("central_dir_offset", c_uint32),
		("comment_len", c_uint16)]

CENTRAL_DIRECTORY_MAGIC = b"PK\x01\x02"
LOCAL_FILE_HEADER_MAGIC = b"PK\x03\x04"
END_CENTRAL_DIRECTORY_MAGIC = b"PK\x05\x06"

# END ZIP FILE STRUCTURES

if len(sys.argv) != 4:
	exit("USAGE: {} cover.jpg data.zip output.zip.jpg".format(sys.argv[0]))

cover_file, data_zipfile, output_file = sys.argv[1:]

data = open(cover_file, "rb")

# BEGIN HACKY JPEG FILE PARSING

soi = data.read(2)
assert(soi == b"\xff\xd8")

while True:
	segtype = data.read(2)
	#print(segtype)
	if segtype == b"\xff\xda": # SOS
		break
	seglen = struct.unpack(">H", data.read(2))[0]
	#print(seglen)
	segdata = data.read(seglen-2)

sos_index = data.tell()-2
data.seek(0)
orig_data = data.read()

# END HACKY JPEG FILE PARSING
# BEGIN HACKY ZIP FILE PARSING

#debugging
def getdict(struct):
	return dict((field, getattr(struct, field)) for field, _ in struct._fields_)

zipdata = open(data_zipfile, "rb").read()

idx = 0
file_datas = []
dirents = []
enddir = None

while idx < len(zipdata):
	magic = zipdata[idx:idx+4]
	if magic == LOCAL_FILE_HEADER_MAGIC:
		start = idx
		h = LocalFileHeader.from_buffer_copy(zipdata[idx:])
		idx += sizeof(h)
		filename = zipdata[idx:idx+h.filename_len]
		idx += h.filename_len
		extra = zipdata[idx:idx+h.extra_len]
		idx += h.extra_len
```

```
                file_data = zipdata[idx:idx+h.comp_size]
                idx += h.comp_size

                file_datas.append(zipdata[start:idx])

                #print(getdict(h))
                print("File {} is {} bytes (compressed)".format(filename, h.comp_size))
        elif magic == CENTRAL_DIRECTORY_MAGIC:
                h = CentralDirectory.from_buffer_copy(zipdata[idx:])
                idx += sizeof(h)
                data_start = idx
                filename = zipdata[idx:idx+h.filename_len]
                idx += h.filename_len
                extra = zipdata[idx:idx+h.extra_len]
                idx += h.extra_len

                dirents.append((h, zipdata[data_start:idx]))

                #print(getdict(h))
                print("Local file header for {} is at offset {}".format(filename, h.local_header_offset))
        elif magic == END_CENTRAL_DIRECTORY_MAGIC:
                h = EndCentralDirectory.from_buffer_copy(zipdata[idx:])
                idx += sizeof(h)

                enddir = h

                print(getdict(h))
        else:
                print("UNKNOWN MAGIC: {}".format(magic))
                break

# END HACKY ZIP FILE PARSING
# BEGIN PACKING ZIP SECTIONS
#TODO: more intelligent chunk packing


value = b""
value_offset = sos_index + 162

for i, file in enumerate(zip(file_datas, dirents)):
        data, dirent = file
        chunk_size = 65521 if i else 65376
        if len(data) > chunk_size:
                exit("Oops, one of the files is too big")
        dirent[0].local_header_offset = value_offset
        value += data + b"A"*(chunk_size-len(data))
        value_offset += chunk_size + 18

enddir.central_dir_offset = value_offset

for dirent in dirents:
        value += bytes(dirent[0]) + dirent[1]

value += bytes(enddir)


# END PACKING ZIP SECTIONS
# BEGIN ICC PROFILE INSERTION

icc_header = b"<!--\x04\x00\x00\x00mntrRGB XYZ \x07\xe1\x00\x07\x00\x07\x00\r\x00\x16\x00\x20acsp" + b"\0"*28 + b"\x00\x00\xf6\xd6\x00\x01\x00\x00\x00\x00\xd3-" + b"\0"*(4+16+28)
table_ents = [
        (b"lmao", value)
]
icc_table = struct.pack(">I", len(table_ents))
icc_table_data = b""
i = 128+4+len(table_ents)*12
for tag, value in table_ents:
        icc_table += struct.pack(">4sII", tag, i, len(value))
        i += len(value)
        icc_table_data += value
icc_table += icc_table_data

icc_data = struct.pack(">I", 4+len(icc_header)+len(icc_table)) + icc_header + icc_table
nchunks = len(icc_data)//0xFFEF + 1

new_file = orig_data[:sos_index]

for i in range(nchunks):
        start = i*0xFFEF
        end = (i+1)*0xFFEF
        chunk = icc_data[start:end]
        #print(hex(len(chunk)))
        new_file += b"\xff\xe2"
        new_file += struct.pack(">H", len(chunk)+16)
        new_file += b"ICC_PROFILE\x00" + struct.pack("BB", i+1, nchunks)
        new_file += chunk

new_file += orig_data[sos_index:]

# END ICC PROFILE INSERTION

with open(output_file, "wb") as f:
        f.write(new_file)
```