Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

```
00000000: 50 4B 03 04 14 00 02 08|00 00 72 01 38 4C 90 E8  | PK........r.8L.č
00000010: 5B 00 66 00 00 00 66 00|00 00 17 00 00 00 50 72  | [.f...f.......Pr
00000020: 69 6E 63 69 70 6F 76 C3|BD 4F 62 72 C3 A1 7A 65  | incipovÃ˝ObrÃ¡ze
00000030: 6B 2E 62 6D 70 42 4D 66|00 00 00 00 00 00 00 36  | k.bmpBMf.......6
00000040: 00 00 00 28 00 00 00 04|00 00 00 04 00 00 00 01  | ...(............
00000050: 00 18 00 00 00 00 00 30|00 00 00 00 00 00 00 00  | .......0........
00000060: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 FF FF  | ..............ÿÿ
00000070: FF FF FF FF FF FF FF 00|00 00 00 FF 00 00 FF 00  | ÿÿÿÿÿÿÿ.....ÿ..ÿ.
00000080: FF FF FF 00 00 00 FF FF|FF FF FF FF 00 FF 00 00  | ÿÿÿ...ÿÿÿÿÿÿ.ÿ..
00000090: 00 00 00 FF 00 00 FF 00|FF FF FF 50 4B 01 02 14  | ...ÿ..ÿ.ÿÿÿPK...
```

Společná část pro otázky označené X

Předpokládejte, že jsme si pomocí hex vieweru zobrazili obsah začátku `.ZIP` souboru (archivu) – viz výpis výše v záhlaví tohoto zadání.

Otázka č. 1 (X)

O formátu `.ZIP` souboru víme, že od offsetu `0x001E` obsahuje samotné znaky jména prvního uloženého souboru, ale nevíme, v jakém je text jména formátu (zároveň víme, že od offsetu `0x0035` jsou uložena již data prvního souboru). Dle uvedeného příkladu rozhodněte a své rozhodnutí detailně zdůvodněte, jaký má asi textový řetězec jména souboru formát a v jakém kódování je nejspíše uložen. Variantu kódování určete co nejpřesněji.

Otázka č. 2 (X)

Předpokládejte, že v zobrazeném souboru je mimo jiné od offsetu `0x0035` uložený nekomprimovaný obrázek ve formátu `.BMP` (specifikace formátu je v příloze). Nakreslete, jaký obrázek je v tomto `.BMP` souboru uložen – tj. nakreslete pixelovou mřížku (obdélníkovou matici), a pro každý uložený pixel určete jeho barvu (čtvereček pixelu touto barvou vybarvěte, nebo do něj napište označení barvy: K = black, R = red, G = green, B = blue, W = white).

Otázka č. 3

O formátu `.ZIP` kromě informací z *otázky 1* víme, že na offsetu `0x0016` je uložené 32-bitové celé bezznaménkové číslo reprezentující velikost 1. uloženého souboru v bytech, pak následuje jméno 1. uloženého souboru (viz řešení *otázky 1*), a hned za jménem souboru jsou jeho data. Napište v Pascalu program, který načte do paměti obsah libovolného `.ZIP` souboru odpovídajícího výše uvedenému formátu, a za předpokladu, že jméno prvního souboru uloženého v `.ZIP` archivu končí „`.bmp`“ (bez uvozovek), tak bude předpokládat, že tento 1. soubor uložený v takovém `.ZIP`u je ve formátu `.BMP` (specifikace viz příloha), a pro tento 1. uložený `.BMP` soubor zobrazí na standardní výstup jeho šířku a výšku v pixelech.

Otázka č. 4

Jaký je rozdíl mezi textovými a binárními soubory? Bylo by možné nějaký textový soubor otevřít pomocí Pascalového typu souboru `file of byte`, a číst z něj data pomocí procedury `BlockRead`? Pokud ano, tak to bychom přečetli, a proč? Pokud ne, tak vysvětlete proč.

Otázka č. 5

Naprogramujte v Pascalu funkci `Conv` s níže uvedeným prototypem, která jako svůj argument `flt32` bere 32-bitů dat, která reprezentují **floating-point číslo typu *single*** (typ *single* je 32-bit číslo dle IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bit exponent uložený ve formátu bias +127, poslední bit, tedy `MSb`, je znaménkový). *Nekonečna* jsou reprezentována jako maximální možná hodnota exponentu s mantisou se všemi nulovými bity, hodnoty *Not a Number* jsou reprezentovány jako maximální možná hodnota exponentu a libovolná nenulová mantisa – oba tyto druhy speciálních hodnot má vaše funkce podporovat.
Úkolem funkce je získat z předaného reálného čísla uloženého v `flt32` jeho celou část (před desetinnou čárkou), a tu vrátit včetně znaménka jako celé 32-bitové číslo `longint`. Tedy pokud je v argumentu `flt32` single reprezentace např. hodnoty `3,14`, tak má funkce vrátit hodnotu `3`, nebo pokud je v `flt32` single reprezentace hodnoty `-145,768`, tak má funkce vrátit hodnotu `-145`. Předpokládejte, že hodnota celé části čísla `flt32` se vejde do rozsahu `longint`. Nicméně pokud je argumentem ***+Nekonečno*** nebo *Not a Number*, má funkce vrátit největší kladnou hodnotu reprezentovatelnou v typu `longint`, pokud je argumentem ***-Nekonečno***, má funkce vrátit nejmenší zápornou hodnotu reprezentovatelnou v typu `longint`. Celý výpočet/zpracování zapište **jen s využitím celočíselné aritmetiky** Pascalu (**bez použití Pascal typů a funkcí pro reálná čísla**), a zvažte možnost použít pro výpočet bitové operace podporované v Pascalu. Poznámka: typ `longword` je 32-bitový bezznaménkový (unsigned) integer, `longint` je znaménkový (signed) 32-bit integer.

```
function Conv(flt32 : longword) : longint;
```

Otázka č. 6

Předpokládejte, že implementujete proceduru:

```
procedure ReadLn(var arg : longword);
```

pro standardní knihovnu naší nové varianty jazyka Pascal. Napište její implementaci bez použití jiných variant procedury `ReadLn`. Můžete předpokládat, že máte k dispozici (můžete ve svém kódu volat) API funkce pro práci se soubory jádra nějakého běžného cílového operačního systému. Také můžete využít standardní Pascal funkce pro převod čísel do textové reprezentace a zpět. Typ `longword` je 32-bit unsigned integer.

Otázka č. 7

Předpokládejte následující program, ve kterém je několik chyb, díky kterým se nechová dle našeho zadání – chtěli jsme, aby program zobrazil každé 4 po sobě následující byty z 8 bytového souboru (tj. 5 různých čtveřic bytů) jako 32-bit floating-point číslo, a pro každé takové číslo zobrazil rozsah adres, kde je v paměti uložené. Detailně vysvětlete, proč se tak program nechová, a navrhněte, jak ho opravit.

```
type PSingle = ^single; PByte = ^byte;
var
  buffer : array[0..7] of byte;
  value : PSingle; i, bytesRead : longword;
  f : file of byte;
begin
  Assign(f, 'Data.bin'); Reset(f);
  BlockRead(f, buffer, 8, bytesRead); Close(f);
  value := PSingle(@buffer[0]);
  for i := 1 to 5 do begin
    WriteLn(
      longword(@value), ' to ', longword(@value) + 3,
      ' contain float value of ', value^ );
    value := value + 1;
  end;
end.
```

Otázka č. 8

Předpokládejte, že program z *otázky 7* spustíme na nějakém moderním operačním systému jako Linux nebo Windows. Nakreslete schéma ilustrující, na jaké hlavní logické celky bude „dle účelu“ rozdělený virtuální adresový prostor takového spuštěného procesu (ke každé části napište stručný komentář o jejím významu). Pokud bude mít takový celek obvykle shodné nastavení oprávnění u všech jeho stránek, tak taková nastavená oprávnění stránek uveďte (resp. budou-li mít různé stránky jednoho celku nastavená jiná oprávnění, tak vysvětlete proč).

Otázka č. 9

Vysvětlete v kontextu typické I^2C sběrnice, co to je tzv. ztráta arbitarace (*arbitration lost*)? V jaké situaci k ní může docházet? Nakreslete příklad časového diagramu průběhu signálů na SCL a SDA lince, když dojde ke ztrátě arbitrace v průběhu přenosu adresy slave zařízení.

Otázka č. 10

Předpokládejte, že známe specifikaci varianty virtual machine vycházející z CLR (Common Language Runtime) = standardní VM .NETu. Naše VM je stroj s **harvardskou** a se **zásobníkovou registrovou** architekturou (pro jednoduchost předpokládejte, že všechny registry obsahují **64-bit floating-point** reálná čísla dle IEEE 754, velikost registrového zásobníku je neomezená). Strojový kód pro tuto VM budeme nazývat tzv. CIL kód (Common Intermediate Language). Víme, že v CIL kódu i v samotné virtual machine jsou všechna data uložena jako **little-endian**, a že instrukční sada VM obsahuje minimálně tyto instrukce (všechny mají jednobytový opcode následovaný případnými argumenty):

- `ldc.r8` – load constant, real 8 byte = load ve variantě immediate, kde argumentem instrukce je konstanta jako 64-bitové floating-point reálné číslo
- `ldsfld` – load static field = load z globální proměnné, jejíž adresa je argumentem instrukce
- `stsfld` – store static field = store do globální proměnné, jejíž adresa je argumentem instrukce
- `call` – volání procedury nebo funkce (argumenty se předávají zleva doprava na registrovém zásobníku a **odstraňuje je volaný**, návratová hodnota se předává na vrcholu registrového zásobníku)
- `ret` – návrat z podprogramu (bez explicitních argumentů)
- `add` – sčítání (bez explicitních argumentů)
- `sub` – odečítání (bez explicitních argumentů), pravý implicitní argument (tj. hodnota, která je odečítána) je ten, který je na vrcholu zásobníku
- `mul` – násobení (bez explicitních argumentů)
- `div` – dělení (bez explicitních argumentů) , pravý implicitní argument (tj. jmenovatel) je ten, který je na vrcholu zásobníku
- `neg` – unární operace negace (bez explicitních argumentů)
- `br` – nepodmíněný skok (branch)
- `bne` – podmíněný skok branch if not equal to: instrukce odebere ze zásobníku dvě hodnoty, a pokud se druhá odebraná nerovná první odebrané, tak se provede skok na adresu, která je argumentem instrukce.

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu, který jsme ze spustitelného souboru obsahujícího CIL kód disassemblovali do assembleru CILu:

```
0x20580000  ldsfld [0x20600008]
0x20580005  ldsfld [0x20600008]
0x2058000a  mul
0x2058000b  ldc.r8 4.0
0x20580014  ldsfld [0x20600000]
0x20580019  mul
0x2058001a  ldsfld [0x20600010]
0x2058001f  mul
0x20580020  sub
0x20580021  stsfld [0x20600018]
0x20580026  ldsfld [0x20600018]
0x2058002b  ldc.r8 0.0
0x20580034  bne    0x20580052
0x20580036  ldsfld [0x20600008]
0x2058003b  neg
0x2058003c  ldc.r8 2.0
0x20580045  ldsfld [0x20600000]
0x2058004a  mul
0x2058004b  div
0x2058004c  stsfld [0x20600020]
0x20580051  br     0x20580078
0x20580052  ldsfld [0x20600008]
0x20580057  neg
0x20580058  ldsfld [0x20600018]
0x2058005d  call   0x20581000
0x20580062  add
0x20580063  ldc.r8 2.
0x2058006c  ldsfld [0x20600000]
0x20580071  mul
0x20580072  div
0x20580073  stsfld [0x20600020]
0x20580078  ret
```