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).

Otázka č. 1

Předpokládejte, že v běžném programu v Pascalu voláme standardní proceduru `Write` s parametrem `42`, která způsobí, že po přeložení a spuštění aplikace pod OS Linux se hodnota `42` vypíše na obrazovku. Kde je procedura `Write(x : Integer)` naimplementována? Jak asi vypadá její implementace a implementace všech jí případně volaných funkcí, které také ještě poběží v uživatelském režimu procesoru, pokud bychom je zapsali v Pascalu?

Otázka č. 2

Předpokládejte, že programujeme nějaký moderní operační systém s podporou pro preemptivní přepínání vláken.
V kernelu tohoto OS programujme API funkci

```
function ReadKey : char;
```

která má počkat na stisk klávesy na klávesnici, a poté vrátit kód stisknuté klávesy. Napište v Pascalu nejvhodnější a co nejkompletnější implementaci takové funkce, pokud má komunikovat se standardním PC AT řadičem klávesnice: řadič generuje `IRQ 1` při každém stisku libovolné klávesy, kód naposledy stisknuté klávesy vrací v 8-bit R/O registru memory-mapped na adrese `0x60`.

Společná část pro otázky označené X

Předpokládejte, že máme počítač s variantou 32-bitového little-endian procesoru Intel Pentium. Procesor má obecnou registrovou architekturu, a mimo jiné 7 obecných 32-bitových registrů `EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`, `EBP`, dále má 32-bitový registr `ESP` (stack pointer na poslední použitý byte), 32-bitový registr `EIP` (program counter), a příznakový registr s běžnými příznaky. ISA obsahuje instrukce `MOV` (má 3 varianty: load, store, přesun mezi dvěma registry), `PUSH`, `POP`, `ADD` (sčítání bez přenosu), `SUB` (odečítání bez přenosu), `IMUL` (násobení), `DIV` (celočíselné dělení se zbytkem: výsledek operace dělení se vždy nachází v registru `EAX`, zbytek po dělení je v registru `EDX`), `JE` (Jump if Equal), `JAE` (Jump if Above or Equal), `CMP` (compare), `JMP`, `CALL`, a `RET` s běžnou sémantikou. Instrukce `MOV`, `ADD`, `SUB`, `IMUL`, `DIV`, `CMP` mají dva argumenty (maximálně jeden smí být typu adresa – viz níže), `PUSH`, `POP`, `JE`, `JAE`, `JMP`, `CALL` mají jeden argument, instrukce `RET` je bez argumentů. Argumentem uvedených instrukcí může být immediate (kde to dává smysl), libovolný obecný registr nebo registr `ESP`, nebo adresa (formát viz níže). Intel syntaxe assembleru pro uvedený procesor je následující: cílový operand instrukce je vždy nejvíce vlevo; hodnota v hranatých závorkách je operand typu adresa, tj. např. `[imm/reg]` znamená hodnotu operandu na adrese dané hodnotou `imm/reg`; operand typu `[reg + imm]` znamená hodnotu na adrese spočítané jako výsledek výrazu v hranatých závorkách, kde `imm` je znaménkový immediate, `reg` je obecný registr nebo `ESP`.
V Pascalu jsme napsali program, který jsme vhodným překladačem přeložili pro uvedený procesor a pro operační systém Windows XP 32-bit. Po spuštění programu na uvedeném systému jsme pomocí debuggeru od adresy `0x009644A0` disassemblovali část dat z adresového prostoru procesu, která odpovídá proceduře `DoSomeMagic`, viz níže (víme, že použitý překladač Pascalu používá variantu Cčkové volací konvence, kdy jsou argumenty funkce ukládány na zásobník zprava doleva, volaný musí volajícímu zachovat minimálně obsah registrů `EBP`):

```
009644A0 push ebp
009644A1 mov  ebp,esp
009644A3 sub  esp,4
009644BE mov  [ebp-4],00000000h
009644C5 mov  eax,[ebp+8]
009644C8 cmp  [eax],000000FFh
009644CE je   0096451B
009644D0 mov  eax,[ebp-4]
009644D2 imul eax,0000000Ah
009644D4 mov  ecx,[ebp+8]
009644D7 add  eax,[ecx]
009644D9 mov  [ebp-4],eax
009644DC mov  eax,[ebp-4]
009644DF cmp  eax,[ebp+10h]
009644E2 jae  009644EF
009644E4 mov  eax,[ebp+0Ch]
009644E7 mov  [eax],00000000h
009644ED jmp  00964507
009644EF mov  eax,[ebp-4]
009644F4 div  eax,[ebp+10h]
009644F7 mov  ecx,[ebp+0Ch]
009644FA mov  [ecx],eax
00964504 mov  [ebp-4],edx
00964507 mov  eax,[ebp+0Ch]
0096450A add  eax,4
0096450D mov  [ebp+0Ch],eax
00964510 mov  eax,[ebp+8]
00964513 add  eax,4
00964516 mov  [ebp+8],eax
00964519 jmp  009644C5
0096451B mov  eax,[ebp+0Ch]
0096451E mov  [eax],000000FFh
00964527 mov  esp,ebp
00964529 pop  ebp
0096452A ret
```

Otázka č. 3 (X)

Pokud hlavní program uvedené aplikace vypadá takto:

```
var
  src : array[0..127] of longword;
  dst : array[0..127] of longword;
begin
  DoSomeMagic(@src[0], @dst[0], 0);
  Writeln('Done');
end.
```

tak po spuštění na příkazovém řádku aplikace vypíše:

```
  Runtime error 200 at $009644F4
```

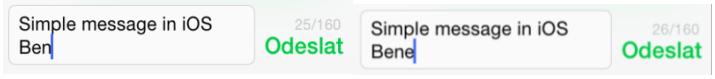
a skončí (bez vypsání textu `Done`). Detailně popište a vysvětlete, kdo a jakým způsobem pozná, že instrukce na adrese `009644F4` dělí nulou, a jaký kód (a proč) se bude vykonávat po neúspěšném provedení instrukce `div`, a jak bude dosaženo ukončení programu.

Otázka č. 4 (X)

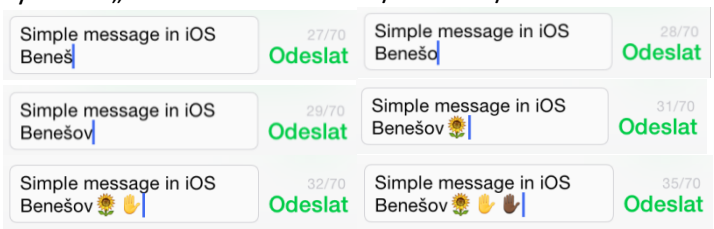
Napište v Pascalu bez použití inline assembleru kód procedury `DoSomeMagic` (i s deklarací), která by mohla být běžným překladačem přeložena do kódu disassemblovaného od adresy `0x009644A0`.

Otázka č. 5

Na telefonu iPhone 5 s iOS 8.4.1 (interní označení 12H321, vydaný k 13. srpnu 2015) jsme ve standardní aplikaci pro posílání SMS zpráv napsali část takové zprávy – v pravé části je zobrazeno, že jsme použili 25 z celkových 160 znaků podporovaných jednou SMS zprávou. Dále jsme dopsáním písmena „e“ zvýšili počet použitých znaků na 26 ze 160:



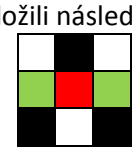
Po zapsání písmena „š“ se zobrazení změní na 27 ze 70 znaků (tedy se pro celou zprávu změnilo kódování všech znaků ve zprávě – jak ukazuje i zapsání dalších znaků „o“ [stav 28/70] a „v“ [stav 29/70]). Navíc se po zapsání symbolu „*slunečnice*“ změní stav skokově z 29/70 na 31/70, po napsání symbolu „*dlaň*“ z 31/70 na 32/70, a po napsání symbolu „*černošská dlaň*“ ze 32/70 na 35/70:



Vysvětlete a zdůvodněte, jaké se asi používá kódování pro 160 znakové SMS zprávy, a jaké pro 70 znakové SMS zprávy. Dále vysvětlete, proč jeden symbol „*slunečnice*“ zabral ve skutečnosti dva znaky v 70 znakové zprávě, a proč symbol „*černošská dlaň*“ zabral dokonce znaky tři? Vysvětlete, a navrhněte jaký kód má asi symbol „*slunečnice*“ a jak je asi zakódován symbol „*černošská dlaň*“ (kód těchto symbolů nemusíte uvést přesně, ale uveďte takové hodnoty, které nejsou v rozporu s výše uvedeným chováním editoru SMS).

Otázka č. 6

Navrhněte co nejjednodušší formát binárních souborů určených pro ukládání nekomprimovaných bitmapových obrázků, tj. popište i to, jak bude vypadat hlavička takových souborů, tj. jaká data a jak tam budou uložena. Váš formát by měl data ukládat standardním způsobem běžným u jiných podobných formátů souborů (můžete se inspirovat např. formátem BMP). Je potřeba, abyste podporovali obrázky ve formátu 16-bit RGB (5 bit na R, 6 bit na G, 5 bit na B), a s libovolným rozměrem (obrázky ale nebudou nikdy větší než 4K, tj. 4096 x 2160 pixelů). Zapište v šestnáctkové soustavě kompletní obsah souboru ve vašem formátu, pokud bychom do něj uložili následující 3 x 3 pixel obrázek:



Otázka č. 7

Naprogramujte v Pascalu program, který jako 1. argument na příkazové řádce – dostupný jako stringový výsledek běžné Pascal funkce `ParamStr(1)` – dostane textový řetězec se jménem binárního bitmapového souboru s formátem, který jste si navrhli v otázce 6. Program má vypsat na obrazovku (standardní výstup) počet čistě zelených pixelů, které jsou v obrázku obsaženy (tedy např. pro výše uvedený obrázek by vypsal číslo 2).

Společná část pro otázky označené Y

Předpokládejte, že máme čistě naformátovaný souborový systém FAT12 používající tabulku FAT – kde, jeden záznam ve FAT má 12 bitů, hodnota `0` reprezentuje volný cluster, konec souboru reprezentuje maximální hodnota. Souborový systém je na disketě se `64 B` sektory, velikost jednoho clusteru jsou 2 sektory. První cluster použitelný pro data souborů (tj. 1. datový cluster) je označený číslem `1`, a odpovídá mu první záznam ve FAT. Pro obsah kořenového adresáře jsou napevno vyhrazeny 4 clustery před 1. datovým clusterem – **pozor:** kořenový adresář je u FAT12 jediný soubor, který má pevný začátek, a veškeré jeho clustery jsou předalokovány kontinuálně za sebou a **nejsou** spravovány FAT tabulkou. Pro každý adresář platí, že jeden záznam adresářové položky má velikost 32 bytů.

Otázka č. 8 (Y)

Zapište v šestnáctkové soustavě (a zdůvodněte) konečný obsah prvních 16 záznamů FAT tabulky po provedení následujících operací (předpokládejte, že pokud je potřeba další volný cluster pro data souboru, tak se v souborovém systému vybere první volný datový cluster s nejnižším číslem; operace „zápis `N` bytů do `X`“ se chápe jako připsání `N` bytů za poslední byte souboru `X`):

1) Vytvoření prázdného adresáře `/DIR`
2) Vytvoření 8 prázdných souborů `/DIR/A1.TXT` až `/DIR/A8.TXT`
3) Vytvoření prázdného souboru `/DIR/B.TXT`
4) Zápis `1 KiB` do `/DIR/B.TXT`
5) Smazání souborů `/DIR/Ai.TXT` pro všechna sudá `i`
6) Zápis `256 B` do `/DIR/B.TXT`
7) Vytvoření 10 adresářů `/DIR/D1` až `/DIR/D10`

Otázka č. 9 (Y)

Předpokládejte, že v Pascalu programujeme aplikaci, která má v následující proměnné `fat` načtená data z 8 clusterů, které v nějaké instanci souborového systému FAT12 reprezentují kompletní obsah jedné FAT tabulky. Napište kód procedury `PrintClusters`, která na obrazovku (standardní výstup) vypíše seznam čísel všech clusterů zabraných nějakým souborem – číslo prvního clusteru zabraného daty takového souboru je předáno v argumentu `first`:

```
var fat : array[0..64*2*8-1] of byte;
procedure PrintClusters(first : longword);
```

Otázka č. 10

Za předpokladu, že typ *single* je 32-bit floating point číslo dle IEEE 754 (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), napište, co vypíše následující program po přeložení a spuštění na little-endian procesoru Intel Pentium:

```
type PLongword = ^longword;
var s : single; p : PLongword;
begin
  s := -0.078125; p := PLongword(@s);
  Writeln(IntToHex(p^, 8));
end.
```