Write your answers to the special response sheet you received (with your name and photograph). If you are using more than single sheet of paper for your answers, then mark each sheet with its number / total number of sheets you will hand over.

Task 1

Assume we have the following function implemented in Pascal programming language:

```
function ToFixedPoint(
      value : longint; exp : longint
) : longint;
```

The function arguments together represent a real number `x`, where `x` is defined as: `x = value * 10`^{exp}. The function returns value of `x` as a fixed-point real number in a 10+22 format. In hexadecimal format write down a return value of `ToFixedPoint` when called with the following arguments (representing a real number `11.53125`):

```
… := ToFixedPoint(1153125, -5);
```

Assume the type `longint` is a Pascal standard 32-bit signed integer number.

Task 2

Assume you are implementing a procedure `Sleep(s : Longint)` as part of a multithreaded operating system (supporting parallel execution of many threads from different applications). The goal of the procedure is to postpone execution of further calling thread instructions by at least `s` seconds. There are several feasible variants of how to implement such procedure – choose the one best suited for the OS described above. Write down a pseudo code for the Sleep procedure as well as all other parts of the OS responsible for the overall behavior of the "sleeping" operation (focus only on code directly responsible for any operation necessary during the whole sleeping sequence of the calling thread). Assume your code will always run on a platform providing you with all necessary devices and controllers.

Task 3

Implement the following function `TextLength` in Pascal:

```
type
  PUtf16 = ^word;
function TextLength(text : PUtf16) : word;
```

where: the argument `text` is a pointer to a null-terminated string in UTF-16 encoding. The function `TextLength` should return a number of graphemes (i.e. "complete characters" including all of their diacritics) the string `text` is composed of (e.g. for any input string representing text `Říp` in Czech the function has to return value 3). Assume the size of type `word` is 2 bytes. You can use reasonable basic utility functions, that an RTL might provide, however for each such case describe the function/procedure required (its prototype and behavior). **Important note:** Providing support for UCS-2 is not enough – you have to provide support for full UTF-16 encoding!

Task 4

Assume you have a clearly formatted file system using a FAT to manage files, where: every FAT entry is 12 bits long, value 0 represents a free sector, end of file is represented by maximal entry value. The file system is located on a hard drive with 512 B sectors; size of 1 cluster is exactly 1 sector. First sector/cluster available for file data (called 1st data sector) has assigned number 1, and is represented by the first entry in the FAT. The file system supports only a single directory (root directory), for which exactly 4 sectors before the 1st data sector are reserved. Every directory entry is exactly 32 B long. Every directory entry contains the following: 11 B for a file name, 2 B for a number of first sector, 4 B for length (in bytes).

Draw and describe contents of the first 16 FAT entries just after execution of all the following commands (if a new data sector needs to be allocated, assume to choose always the first unallocated sector with the lowest number; an operation of "write N bytes to X" should be interpreted as writing [appending] N bytes after the last byte of file X):

1) Create new empty file A.TXT
2) Create new empty file B.TXT
3) Write 500 bytes to B.TXT
4) Write 4 KiB to A.TXT
5) Write 1 KiB to B.TXT
6) Delete file A.TXT
7) Create new empty file C.TXT
8) Write 1 byte to B.TXT
9) Write 512 bytes to B.TXT

Task 5

The following program written in Pascal, when compiled using the Free Pascal compiler, is executable on both Linux OS running on x86 platform and Windows XP OS running on 86 platform. Describe and explain how and why it is possible. Also describe all the steps necessary to get from the original source code to having a suitable executable file. Don't forget to explain if a single executable file is sufficient in the scenario described above, or if multiple executable files are required.

```
program Power;
var
  m, n : integer;

begin
  ReadLn(n);
  m := 1;
  while n >= 1 do begin
    m := m * 2;
    Dec(n);
  end;
  WriteLn('2 to power of N equals ', m);
end.
```

Task 6

Assume you are implementing an OS function whose responsibility is to send N bytes of data across a local computer network. The function has two arguments it receives from an application: a pointer to first byte of data to be transferred, and value N. The network interface card (NIC) is programmed via *port-mapped I/O*, and the NIC is using DMA with *scatter/gather I/O* support to transfer data to/from main RAM memory. Thus every time a data transfer should be initiated the OS has to provide the NIC with an address of data to be transferred. However, to maximize system throughput we would like to support only scenario where the NIC receives an address of the original buffer in the originating application (i.e. we never want to copy any data to a different memory location). With an assumption **the OS is using paging** manage processes memory explain the following:

a) Describe and explain all actions the OS has to take in order to pass a correct address to the NIC. Provide an example.
b) Can you apply such mechanism to all combinations of source address and N, or are there any addresses and/or Ns not valid for applications to pass to the OS here? Explain and describe all variants.

Task 7

Assume you are designing a sound card for the 16-bit ISA bus (16-bit data bus, dedicated 24-bit address bus, special 16-bit I/O address space, supports bus mastering). The sound card should support playback of uncompressed audio in CD format only (2 channels [stereo], 16-bit samples, 44.1 kHz sampling rate). Propose and define a suitable HCI for such a sound card (you can define any necessary port addresses as well as any other constants to any reasonable value). Your HCI has to have all the following properties:

1) Supports scatter/gather IO.
2) Single "Play" command can be used to start audio playback (command has two arguments: a buffer containing the audio data, and size of the buffer in bytes [you can define a reasonable maximum allowed buffer size]).
3) The sound card has to suitably indicate a state when it is transferring data from main memory. The sound card does not accept (= ignores) all "Play" commands in such a state.

Task 8

An OS supporting multithreading has to suddenly terminate a thread (e.g. as a result of null pointer dereference in that thread) that holds several locked locks. The lock implementation is provided by the OS itself. Explain what the OS should do in such a situation, describe all typical solutions of the problem, and for each of them explain its pros and cons.

Task 9

Describe and explain what a "*sandbox*" is, and where and for what reason it is used.

Task 10

Assume to have PC with a 32-bit PCI bus. There is a GPIO (General Purpose Input/Output) controller connected to the PCI bus. The controller provides access to 32 independent digital output signals/lines (called GPIO). The controller is configured to have a single 32-bit port mapped onto the I/O address space address `0x4567`. Every bit of the port represents state of a single GPIO output signal. The port can be read to receive current state of all the GPIO signals the controller is "transmitting". By writing to the same port we change the state of all the signals controller is "transmitting". Your goal is to implement (in Pascal variant with a typical suitable extension, e.g. Free Pascal) a procedure with the following prototype:

```
procedure Output(b : Longword);
```

where `Longword` is a 32-bit unsigned integer type, and valid value of b argument is any number in range from `0` to `255`.

The goal of the procedure is change in a single moment in time (**at once**) all the GPIO signals in the following way (all IDs are counted from `0`): 2nd and 3rd GPIO to `0`, 20th to 23rd GPIO to values of bits 0 to 3 of the b argument, 28th to 31st GPIO to **negation** of values of bits 4 to 7 of the b argument. All other GPIO lines must remain unchanged!

You can assume your code will always run on a computer with Intel Pentium III microprocessor (32-bit CPU, 64-bit data bus, 36-bit address bus, special 16-bit I/O address space). Assume the processor, the PCI bus, and the GPIO controller all have the same bit and byte ordering. The CPU contains (among others) 4 general 32-bit registers EAX, EBX, ECX, EDX. Its instruction set includes the following instructions:

- `MOV` *`op1, op2`*
  Where only one of *op1* and *op2* is allowed to be an address, *op1* = target (register or address), *op2* = source (register, address [marked by square brackets] or immediate value).
- `IN EAX, EDX`
  Reading 32-bit value into EAX from an address (in EDX) of the I/O address space.
- `OUT EDX, EAX`
  Writing value of EAX into 32 bits in I/O address space given by an address in EDX.

The instruction set also includes instructions for all common unary and binary arithmetic and bit operations – all such instructions follow this format: `unInstr` *`op1`* or `binIntr` *`op1, op2`*, where: *op1* = target (register only), *op2* = 2nd source (register, immediate, or address).

Important note: instructions `IN` and `OUT` of the Intel Pentium III processor have also additional variants with **8-bit, or 16-bit** operand. However, we **cannot** use them in our implementation, even though they look equivalent – e.g. for consecutive 8-bit writes to addresses `0x4567`, `0x4568`, `0x4569`, `0x456A` are not equivalent to a single 32-bit write to address `0x4567`, as they will **not** change state of all the GPIO lines **at once**.