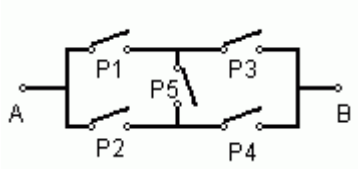
Zadání písemné části zkoušky I.

1. Napište v jazyce Pascal funkci AB pěti logických proměnných P1..P5, která vrací logickou hodnotu odpovídají stavu vodivosti mezi body A a B, pokud její parametry P1..P5 typu boolean mají význam stavu spínačů na obrázku.



2. Co bude výstupem následujícího programu

```
program PQ1;

procedure F(n: integer);
begin
  if n<2 then
    write(n)
  else begin
    F(n-1);
    F(n-2);
  end;
end;

begin
  F(4);
end.
```

3. Nechť indexy v následující matici mají význam kartézských souřadnic bodu v rovnině v milimetrech. V matici hodnot elektrického napětí U[i,j] pak najděte místo největšího spádu elektrického potenciálu. Uvažujte spád pro sousední body ve směru řádků, sloupců i diagonální směry. Předpokládejte, že funkce nebo procedura, která má vrátit výsledek následuje po deklaracích:

```
const M = 50;
      N = 50;

var   U : array [1..N,1..M] of real;
```

Vymyslete hlavičku i tělo procedury či funkce, která vrátí požadované místo největšího spádu elektrického potenciálu v podobě dvou dvojic (i,j). Výše uvedené deklarace neměňte. Spád počítejte jako rozdíl napětí dělený vzdáleností v milimetrech. Úhlopříčka čtverce o hraně 1 mm je sqrt(2) milimetru. V případě, že míst s největší hodnotou spádu je více, vrátí funkce libovolné z nich.

4. Uvažujte následující deklarace

```
type tComplex = record
                  Re, Im : real;
                end;

function MaxAbs(const P : array of tComplex) : tComplex;
begin
  ...
end;
```

Nahraďte tři tečky (...) tělem funkce tak, aby vracela z pole komplexních čísel to, které má největší absolutní hodnotu.

Zadání písemné části zkoušky II.

1. Napište funkci *f* dvou reálných proměnných *x* a *y* , která vrací reálnou hodnotu danou vztahem

$$f(x,y) = \sqrt{\frac{e^x - x(1+\sqrt{1+y^2})}{e^{-x} + x(1-\sqrt{1+y^2})}}$$

2. Co vypíše následujícího program ?

```
program Test;

var a,b: integer;

procedure X(var a,b : integer);
begin
 a := a+b;
 b := a-b;
end;

begin
  a := 1;
  b := 2;
  X(b,a);
  Writeln(a,b);
end.
```

3. Doplňte v následujícím kusu programu vnitřnosti funkce *OpakujeSe* tak, aby vracela logickou hodnotu *True*, právě když se v poli *A* (jejím parametru), opakuje některá z hodnot. Funkce tedy oznamuje, že zobrazení z intervalu celých čísel *tIndex* do intervalu celých čísel *tHodnota* dané polem *A* není prosté.

```
program Test2;

const MaxHodnota = 300;
      MaxIndex   = 200;

type tHodnota = 0..MaxHodnota;
     tIndex   = 1..MaxIndex;

     tPole    = array [tIndex] of tHodnota;

function OpakujeSe(const A : tPole) : Boolean;
begin

end;
```

Zkuste nalézt algoritmus s časem výpočtu lepším než O(MaxIndex^2). Hodnoty konstant jsou uvedeny proto, abyste mohli odhadnout, zda váš algoritmus je pro tyto hodnoty na běžném počítači použitelný, nikoli proto, abyste mohli "optimalizovat" zápis kódu pomocí numericky explicitních hodnot mezí cyklů (for i := 1 to 499 do ....) atp.

4. Uvažujte následující deklarace

```
type tBod2 = record
                  x, y : real;
                end;

function Plocha(const P : array of tBod2) : real;
...
begin
  ...
end;
```

Nahraďte tři tečky (...) deklaracemi a tělem funkce tak, aby vracela plochu konvexního mnohoúhelníku se souřadnicemi vrcholů uloženými v poli P, parametru funkce. Každý z vrcholů je v poli uložen právě jednou, vrcholy jsou v poli uloženy ve stejném pořadí, v jakém tvoří obvod mnohoúhelníka.
Nápověda: vzpomeňte si, že ve třech dimenzích má vektorový součin dvou vektorů velikost rovnou dvojnásobku plochy trojúhelníku, jehož vrcholy tvoří počátek a "koncové body vektorů". Z toho vyplývá, že plochu trojúhelníku lze přímo spočíst jako polynom druhého stupně v kartézských souřadnicích vrcholů:

$$S = \frac{1}{2}|(\vec{r}_1 - \vec{r}_0) \times (\vec{r}_2 - \vec{r}_0)| = \frac{1}{2}|(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)|$$

Konvexní mnohoúhelník obsahuje všechny své úhlopříčky, takže jeho plochu lze spočíst snadno rozdělením na trojúhelníky pomocí všech uhlopříček procházejících nějakým vybraným vrcholem.

Zadání písemné části zkoušky III.

1. Napište funkci *JeSamohlaska,* která vrátí logickou hodnotu True právě když je její parametr typu *char* samohláska.

2. Co vypíše následujícího program ?

```
program Test;

var a,b: integer;

procedure X(var b,c : integer);
begin
 a := a+1;
 b := b+2;
 c := c+3;
end;

begin
  a := 1;
  b := 2;
  X(b,a);
  Writeln(a,b);
end.
```

3. Doplňte v následujícím kusu programu vnitřnosti funkce *ADost* tak, aby vracela logickou hodnotu *True*, právě když se v řetězci *S* (jejím parametru), opakuje některé z písmen více jak dvakrát (tedy třikrát nebo vícekrát).

```
program Test2;

function ADost(const S : String) : Boolean;
...
begin
...
end;
```

Pro slovo 'Nejnemoznejsi' tedy *ADost* vrátí *true* zatímco pro slovo 'Eee' vrátí *false*.

4. Uvažujte následující deklarace

```
const Dim = 7;
      eps = 1E-10;

type tVektor = array [1..Dim] of real;

function SkalSouc(const a,b : tVektor) : real; forward;

procedure OdectiOdXLambdaKratY(var X : tVektor; const Y : tVektor; lambda: real); forward;

function JsouNezavisle(P : array of tVektor) : boolean;
...
begin
  ...
end;
```

Nahraďte tři tečky (...) deklaracemi a tělem funkce *JsouNezavisle* tak, aby vracela *true* právě když je její parametr P tvořen na sobě lineárně nezávislými vektory. Použijte metodu s níž jste se seznámili v hodinách lineární algebry, tzv. Gramm-Schmidtovu ortogonalizaci. Ta spočívá v tom, že k-tý vektor P[k] upravíme přičtením vhodných násobků vektorů P[j], j=0..k-1 tak, aby skalární součin upraveného vektoru P[k] s vektory P[j], j=0..k-1 byl nulový. Protože ale vektory P[j], j=0..k-1 byly již ortogonalizovány a jsou tedy navzájem kolmé, lze odečítání násobku vektorů provést postupně. Pro naše potřeby je výhodné konstruovat bázi ortonormální, kdy každý P[k] po odečtení projekcí ještě vydělíme jeho velikostí. Zjednoduší to ortogonalizaci, protože hlavní příkaz ortogonalizace pak je
OdectiOdXLambdaKratY(P[k] , P[j] , SkalSouc(P[k] , P[j]));
Navíc pokud pokud normalizace provést nejde, tedy norma vektoru po odečtení projekcí na předchozí vektory báze je nula (přesněji |P[k]| je menší než deklarovaná konstanta *eps*), jsou původní vektory lineárně závislé a nelze pokračovat.

Vnitřnosti funkce SkalSouc a procedury OdectiOdXLambdaKratY psát nemusíte.

Poznámka: algoritmicky ale především numericky vhodnější metoda vychází z Gaussovy eliminace s prohazováním řádků i sloupců, ale tu jste v hodinách lineární algebry nejspíš neprobírali.